POLITECNICO DI TORINO

MASTER's Degree in ELECTRONIC ENGINEERING

MASTER's Degree Thesis

From minutes to millennia: enhancing power analysis resistance in AES and ASCON

Supervisors

Prof. GUIDO MASERA

Prof. MATTIA MIRIGALDI

Candidate

MATTIA CASTAGNO

Academic Year 2023/2024

Summary

Cryptographic algorithms are an important pillar in the digital world because they protect the content of sensitive data ensuring security in communication. Their strength lies in the mathematical complexity that underlies them. Thanks to it, the encryption of data is performed in a very simple way, while the reverse operation requires a huge amount of work in the case in which all the parameters used to perform it are not known. This peculiarity ensures strong protection against direct attacks such as brute force attacks or more advanced techniques such as linear or differential cryptanalysis, which would require millennia to recover with certainty the value of the cryptographic key.

However, with the advent of a new class of attacks, known as side-channel attacks (SCA), this security has disappeared, since they no longer aim to discover the key based on the mathematical structure on which they are based, but they exploit weaknesses in the physical implementation of the algorithm, such as execution time or power consumed. The study that has been done in this thesis has the aim of investigating the behavior of two encryption algorithms subjected to power analysis attacks. The first is the Advanced Encryption Standard (AES), which at the time it was designed had as its main focus to be very robust against linear and differential cryptanalysis. This strength, however, makes it very weak against power analysis attacks. ASCON, its lightweight counterpart that has also recently become a standard, on the other hand has been conceived since its creation with the aim of being resistant against this type of attacks. With the right techniques, however, power analysis can still be a threat to this algorithm.

This study, in addition to investigating the vulnerabilities of AES and ASCON, also aims to test a lightweight solution to increase resistance to power analysis. In particular, one of the most critical blocks regarding this type of cryptanalysis, the substitution box (S-box), is replaced with alternative S-boxes. Their choice is made by comparing with the original one of these algorithms their cryptographic properties, such as the confusion coefficient (CC) and the transparency order (TO). However, these metrics bring to light a very important consequence, namely finding the best trade-off between cryptanalytic security and power resistance. The effectiveness of these proposed S-boxes is tested on a physical implementation to evaluate their actual effectiveness in this balancing.

Acknowledgements

"You don't have to understand your feelings completely to know you like something. You don't have to always have figured everything out. You can just feel."
Nick Nelson, Heartstopper

A *Carmela*, una mamma forte, mi hai insegnato il vero significato dell'amore, mostrandomi che si trova nei piccoli gesti, nella pazienza, nella cura reciproca e nella capacità di esserci anche nei momenti più difficili.

A *Roberto*, un papà indispensabile, mi hai insegnato a guardare il mondo con curiosità, a meravigliarmi di fronte alle sue infinite sfaccettature e a cercare sempre di andare oltre l'apparenza, scoprendo nuove prospettive e trovando bellezza anche nei dettagli più semplici.

A *Fabio*, il miglior fratellino con cui crescere, se ripenso ai miei ricordi più felici, ci sei sempre tu, con la tua presenza che rendeva ogni momento speciale. Sei stato il filo conduttore delle risate più autentiche, delle emozioni più intense e dei momenti che porto nel cuore con gratitudine e gioia.

A *Maria*, i tuoi racconti sono una preziosa lezione su chi siamo stati e su chi siamo diventati, un ponte tra il passato e il presente che mi aiuta a comprendere le radici della nostra storia e il percorso che ci ha portati fino a qui. Attraverso le tue parole rivivo emozioni, scopro valori dimenticati e trovo ispirazione per affrontare il futuro con maggiore consapevolezza.

A *Margherita*, hai saputo trasmettermi la passione per il cibo, facendomi scoprire non solo i suoi sapori, ma anche il valore delle tradizioni, delle storie e dell'amore che si nasconde dietro ogni piatto.

Ad *Angelo*, la vita non è stata generosa con te, ma nonostante le difficoltà hai saputo trasmettermi forza, valori e insegnamenti che hanno plasmato il ragazzo che sono oggi. Con il tuo esempio mi hai mostrato cosa significhino il sacrificio, la determinazione e l'amore incondizionato, regalandoci il dono più prezioso: la tua saggezza e la tua dedizione.

Alla *papero*, che mi ha fatto sentire accettato in un mondo dove l'accettazione è difficile da trovare, dove spesso ci si sente giudicati o invisibili. Con la tua presenza, la tua comprensione e il tuo affetto, mi hai dato un rifugio sicuro, dove potevo essere me stesso senza paura di non essere compreso o accolto.

A *Stefano*, il mio neurone, con cui posso ridere delle cose più insignificanti o confrontarmi su argomenti profondi e complessi.

A *Francesca*, che mi sopporta con pazienza nelle mie stranezze, trovando sempre il modo di sorridere insieme nelle piccole cose.

A *Melany*, che ha sempre notizie fresche da condividere, capaci di generare conversazioni che potrebbero durare ore.

A *Giada*, con cui condivido risate, scherzi e complicità che rendono ogni momento più leggero e divertente.

A *Vanessa*, per farmi sempre ridere e per riuscire a rendere ogni momento più divertente e spensierato, anche quando meno te lo aspetti.

Ad *Alessia*, che conosco fin dall'asilo e che è stata al mio fianco in ogni fase del mio percorso: dalle risate e giochi dell'infanzia, alle avventure delle elementari, alle sfide delle medie, fino ai momenti indimenticabili delle superiori. Sei stata una presenza costante e insostituibile, sempre pronta a sostenermi, a condividere i momenti più belli e ad affrontare insieme le difficoltà. Con te, ogni giorno è stato speciale e sono grato per tutto ciò che abbiamo vissuto insieme.

A *Tattoo*, la mia seconda famiglia. Ogni lezione, ogni allenamento, ogni risata e ogni gara insieme hanno contribuito a rendere questo percorso speciale. Con voi ho imparato tanto, non solo sulla danza, ma anche sulla forza dell'amicizia e sull'importanza di condividere passioni e obiettivi.

Per aspera ad astra

Table of Contents

List of Tables

List of Figures

Acronyms

SDA
Side Channel Attack

SPA
Simple Power Analysis

DPA
Differential Power Analysis

CPA
Correlation Power Analysis

AES
Advanced Encryption Standard

IV
Initial Value

ECB
Electronic Codebook

CBC
Chiper Block Chaining

OFB
Output Feedback

CFB
Cipher Feedback

CTR

Counter

AEAD

Authenticated encryption with associated data

NL

Nonlinearity

DU

Differential uniformity

CCV

Confusion coefficient variance

MCC

Minimum confusion coefficient

TO

Transparency order

TVLA

Test Vector Leakage Assessment

SNR

Signal-to-Noise-Ratio

Chapter 1

Cryptography Overview

Since ancient times, from powerful emperors to the humblest artisans, man has sought methods to conceal the secrets contained in written texts. One of the earliest records of this primitive cryptography comes from ancient Mesopotamia, where around 1900 BC a craftsman wrote on a clay tablet the secret formula for producing ceramic glazes using a technique called cuneiform cryptography. This type of encryption was a rudimentary way of hiding information, based more on the manipulation of symbols and their arrangement rather than on a true cipher system. More famous is the Caesar cipher, named after the Roman emperor Julius Caesar who used it, according to the Roman historian Suetonius. Here we move on to a real cipher system, where the encryption of the message is done by shifting each letter by three positions in the alphabet. Although nowadays this type of encryption may seem very easy to break, in Caesar's time it was very strong because it was based on the fact that the attacker was illiterate or poorly educated, making this technique effective. An improvement on this type of cipher came in the 16th century by an Italian named Giovan Battista Bellaso who invented the Vigenère cipher. This technique, which was used until World War I, is based on the Caesar cipher with the addition of a key: in this case the letters are shifted by values defined by the key, a collection of letters that represent numbers based on their position in the alphabet. For example, if the key was KEY the letters in the text are shifted using the values 10, 4, 24, and the pattern 10, 4, 24 is repeated until the entire text is encrypted. The Vigenère cipher at the time it was used was strong enough to encrypt the message securely since most messages only needed to be kept secret for short periods of time, and so whether the text was eventually decrypted did not matter.

After this brief digression on the history of cryptography we can immediately understand what the bases for encrypting are. The main purpose of encryption is to make data unintelligible in order to make it confidential. To do this, it uses an algorithm (which is called a *cipher*) and a secret value (called a *key*) to transform

a text that is to be secreted (called a *plaintext*) into another text with another format (called a *ciphertext*) that no one, except those who know the key, can read. However, a cipher is made up of two functions: *encryption*, which is what has just been described, transforms the plaintext into a ciphertext, and *decryption*, which is the reverse algorithm, which transforms the ciphertext into a plaintext.

From this we can understand that not all algorithms can be used to encrypt a message. In fact, an algorithm, in order to be used, must have exactly one inverse, it must perform what is called a *permutation*.

Example. For example, a substitution that transforms the letters A, B, C and D respectively into C, D, D and A is not a permutation because both B and C are mapped to D.

However, not all permutations can be used because a fundamental thing for a cipher is that the result must be secure. In order to be secure, the permutation must satisfy three criteria:

- to make the permutation secret as long as the key is secret, the permutation must be determined by the key;
- using different keys the permutation must give different results, otherwise there would be few possibilities to be found when trying to decrypt without the key;
- once the encryption is done, the result must not present any patterns, otherwise it would make it predictable to an attacker, therefore less secure.

A secure permutation is a necessary but not sufficient condition for building a good cipher. In fact, ciphers, in addition to the permutation, have another main component, the *operating mode*. To be secure, the operating mode of a cipher must ensure that, for the same letters in the plaintext, the ciphertext must have a different permutation. In the case of the Vigenère cipher, if one had the BANANA plaintext of 6 letters while the key was only 2, the last two pairs of letters would be encrypted with the same pattern, and therefore an attacker would learn something about the message. If instead one had a key of 6 letters this problem would be avoided.

So, to build a good cipher, these two conditions must be satisfied, ensuring that an attacker cannot learn anything about the message except its length [1].

The historical examples that were made previously used the same key for decryption as for encryption. This type of encryption is called *symmetric encryption*.

There are instead ciphers that use a different key to decrypt, which are called *asymmetric encryption*, or public-key encryption. These types of ciphers use a pair of mathematically linked keys: the public key, which can be shared with anyone, and the private key, which must remain secret. If a message is encrypted with the public key of a recipient, only the corresponding private key can decrypt it, while if a message is encrypted with the private key, anyone with the public key can decrypt it.

There are advantages and disadvantages to both types. Symmetric encryption, compared to asymmetric encryption, has the advantage of being faster and more efficient, requiring less computational resources, but asymmetric encryption has greater security in the distribution of keys since the public key can be exchanged with everyone, also allowing to verify the authenticity and integrity of the message through the digital signature. In reality they are often used together, as for example in the HTTPS connection where asymmetric encryption is used to exchange a symmetric session key which is then used to encrypt the rest of the communication.

So in light of all this, the main characteristic for a cryptographic algorithm to be secure is the following. Let's take for example two people, Mattia and Stefano, who want to communicate securely, and to do so they use a cipher, whose key is known only to the two of them. In this case, the cryptosystem must have the encryption and decryption operations for them that are computationally simple, but in the case where there is a third person, Fabio, who does not know the key, the decryption problem for him must be computationally impossible. This concept is what is called a *one-way function*, that is, a function that is easy to calculate in one direction but extremely difficult to invert if specific knowledge is missing, such as the secret key. This type of difficulty is crucial for the security of cryptographic systems [2].

This ability to be a good one-way function can be mathematically calculated through two cryptographic properties of the algorithm: *nonlinearity* and *differential uniformity*. Nonlinearity was presented in 1948 by Claude Shannon in his famous article "Communication Theory of Secrecy Systems" [3]. In this work Shannon introduces fundamental concepts for modern cryptography, such as *confusion* and *diffusion*, two very important properties for building a resistant cryptographic algorithm. It lays the foundation for the use of nonlinearity as a defense against cryptanalysis attacks, which has been developed over the years and used in modern algorithms.. Nonlinearity therefore is a metric that measures how much a function differs from a linear function, which is easily invertible. A highly nonlinear function is more difficult to analyze, because each bit of the output has a complex, nonlinear relationship to the input.

The concept of differential uniformity, on the other hand, was introduced in 1993 by Kaisa Nyberg in [4]. The paper shows that to have good resistance against differential cryptanalysis it is sufficient that for every fixed non-zero input difference

to a function there is no output difference with high probability. So in simple terms it is required that the distribution of outputs has a uniform upper bound, and the more uniform it is, the more difficult it will be for an attacker to infer a relationship between inputs and outputs. This distribution is the differential uniformity metric. The lower this metric, the more secure a cryptographic function is considered since it changes unpredictably with small changes in the input.

In summary, therefore, an algorithm to be a good one-way function must have a high nonlinearity and a low differential uniformity. In this way Mattia and Stefano can communicate securely with each other without Fabio being able to read what they are writing, unless he manages to discover the value of the secret key. Nevertheless, if these properties are wisely designed, this possibility is very unlikely to happen, because the algorithm is considered secure against cryptanalysis, in particular brute-force attacks.

However, not all attacks rely solely on the mathematics of the algorithm. Some types of attacks can steal sensitive content by tracing an unexpected path in the digital system exploiting the system implementation or identifying properties in the implementation to break the security. These are called physical and *side-channel* attacks, and can be classified into *invasive* and *non-invasive* attacks. Invasive attacks interfere with and modify system internals, examples of which are micro-probing and reverse engineering. These types of attacks are hard to launch because they require expensive equipment. Non-invasive attacks, on the other hand, do not require opening the device. They only exploits externally available informations such as running time and power consumption, so they are very cheap and scalable.

One of the first official information relating to SCA attacks is described by P. Wright in his book [5] and can be dated back to the 1950s. In 1956, the tension between Britain and Egypt were rising rapidly, and MI5 knew how to decode the messages but were unable because they did not have computers powerful enough to do so. The only thing they knew was the type of cipher machine they used, the Hagelin, manufactured by the Swiss firm Crypto AG. Wright then had the idea of borrowing one to experiment with.

This Hagelin was a keyboard machine, with tape containing the enciphered message leading out from one side. The principle of the machine was simple. Seven rotating wheels, powered by switched currents, automatically substituted mechanically produced random figures for whatever was typed into the machine. Every morning the cipher clerk operating a Hagelin inside an embassy reset the wheels before beginning transmissions.

At this point, the SCA attack comes into play. In order to decipher the message it was necessary to know the "core position" of the machine. Then they installed Special Facilities in the embassy's phones and used their microphones to capture the sound of the cipher machine. Then using highly sensitive microphones at various distances from the borrowed machine and connecting their outputs to an

oscilloscope, they managed to transform the recorded sound into a visual reading. By comparing the reading they took over the phone in the embassy and the set of readings taken on the test machine they could figure out the configuration of at least three out of seven wheels, enough to calculate the new configuration.

This new technique became known by the code word ENGULF, and this operation enabled the Britain to read the Egyptian cipher in the London Embassy throughout the Suez Crisis.

However, the first seminal works related to SCA attacks in the cryptography research community are due to Paul Kocher [6][7][8]. SCA do not use vulnerabilities in the design or algorithm to carry out malicious activities, but rather exploit the design or algorithm itself during execution to get to what is needed. These attacks work because there is a direct correlation between what the machine is doing and the physical measurements during execution, and everything is related to the key that is being used to encrypt the content.

There are numerous ways to implement a side-channel attack, and their implementation varies depending on the type of attack being launched and the device being targeted. A good attacker is one who is able to understand right away, depending on the algorithm or device he wants to break, which type is most suitable and most efficient, so as not to spend days or weeks trying to recover the secret key for encryption. Analyzing in detail the various types of non-invasive side-channel attacks, it is possible to classify them into the following categories:

- **Timing attack**. In 1996 Paul Kocher showed in [6] that by making accurate measurements of a system's execution time, an attacker can discover sensitive information such as an algorithm's encryption keys. These types of attacks are called timing attacks. Every logical operation in a digital system has a time delay from when the input is submitted and the output is provided, and the duration of the execution can differ from input to input. This is due to the fact that for optimization reasons some operations can be bypassed, or to the presence of branches or conditional statements, but also and above all to the different execution time of the instructions in the processor. This vulnerability can be exploited by an attacker very easily because the only thing he has to do is a good time measure, making this type of attack the easiest to launch among SCA.

- **Fault injection**. This type of attack is one of the riskiest in terms of system security compared to other side-channel attacks because not only can it be used to discover the cryptographic key of an algorithm, but an attacker can gain complete control of the system bypassing security checks. The basic idea of this type of attack is very simple, namely to make the hardware run outside of the normal operations it would perform by injecting faults into the device. The less precise the fault injection, the more unexpected and random

the effects in the device will be, which means that only some of them can be exploited. To ensure that the attack succeeds in a reasonable amount of time, an attacker tries to minimize this number of fault injection attempts.

- **Electromagnetic analysis**. Electromagnetic analysis has been a known threat since the 1980s, when Wim Eck published [9] in which he explained how it was possible to reconstruct the contents of a video display unit just by collecting the electromagnetic radiation it emitted. So these attacks are based on capturing the electromagnetic radiation of a device to find sensitive information. Even if on paper they may seem very simple to carry out, in reality great knowledge of the layout of the chip is required to carry out the attack in order to isolate the specific region where the electromagnetic radiation is to be measured.

- **Power analysis**. Power analysis is one of the best performing side-channel attacks because by simply inserting a resistor between the power supply and the device and acquiring the power consumed by it during its use, excellent results can be obtained in a short time. This is due to the fact that, as with timing attacks with time, the different operations performed have different power consumption, and this correlation is exploited by a potential attacker. This type of attack is explained in more detail in the following chapters.

This thesis work has as its first objective to find solutions that do not significantly impact the area and performance of the cryptographic algorithm but at the same time improve its resistance to side-channel attacks. In particular, two important ciphers in the world of cryptography, *AES* and *ASCON*, have been studied, modifying a fundamental block in encryption that adds nonlinearity and confusion, the *Substitution Box*. As a second objective, instead, we wanted to examine the *cryptographic properties* of this layers, going to see if the mathematical results that are obtained have a confirmation on a physical implementation, and therefore are not only theoretical metrics.

The structure of the thesis is as follows. In Chapter 2 the AES algorithm is described, analyzing its internal structure, strengths and vulnerabilities with respect to cryptanalysis. The same approach is adopted for the more recent ASCON algorithm in Chapter 3. In Chapter 4 side-channel attacks based on power analysis are explored in depth. Subsequently, in Chapter 5, countermeasures for these two types of algorithms are proposed, together with the results obtained. Finally, in Chapter 6 the conclusions drawn from these results are presented.

Chapter 2

Advanced Encryption Standard (AES)

2.1 Introduction

A special type of ciphers, which includes the most widely used algorithms today, are *block ciphers*, which combine a core algorithm that works on blocks of data with a mode of operation for processing sequences of data blocks. This cipher has the same security goals that were explained before in Chapter 1, so it should be a pseudorandom permutation, which means as long as the key is secret an attacker should not be able to compute the ciphertext from any input, and there should be no pattern that attackers can exploit to derive the key.

A block cipher is characterized by two fundamental values on which its security depends: the *block size* and the *key size*. The block size mustn't be too large so that both the ciphertext length (the larger the block, the more overhead to compute the ciphertext) and the memory footprint (so that it can fit in the registers of most CPUs) are minimized. For this reason, most block ciphers have either 64-bit or 128-bit blocks.

The structure of a block cipher is very simple. It is composed of *rounds* (basic transformations that are easy to implement, very weak alone but strong if there are many of them) that are repeated several times to form the algorithm. Each round needs to have an inverse so that the recipient can calculate the plaintext again. The functions of each round are identical algorithms that use different round keys, and therefore each round will have a different output if fed with the same input.

The round keys are derived from a main key using an algorithm called a *key schedule*, and must all be different otherwise this would result in a less secure block cipher, making the algorithm more vulnerable to slide attacks. These types of attacks look at two pairs of plaintext and ciphertext (P_1, C_1) (P_2, C_2), and when

two rounds are identical the relation between two plaintexts $P_2 = R(P_1)$ implies the relation $C_2 = R(C_1)$ with respect to their respective ciphertexts. Thus, knowing the input and output of a single round often helps to recover the key. Another advantage of using round keys is protection against side-channel attacks, in fact if the function to find the key for a round from the main key is not invertible, then if an attacker managed to find the key for that round he would not be able to derive the main key.

There are two main techniques for building a round. The first is the *Feistel scheme* which was designed in the 70s by IBM engineer Horst Feistel. It works on two 32-bit blocks L and R, and the computation that is done with these blocks is $L \oplus F(R)$. After that, L and R are swapped and the next round is started.

The second technique is the *substitution-permutation network*. In cryptography there are two very important properties, *confusion* and *diffusion*. Confusion means that the input undergoes complex transformations, while diffusion means that these transformations have the same weight on all the bits of it. In this technique, confusion and diffusion take the form of substitution and permutation.

S-boxes, or substitution boxes, are often used to perform the substitution, and they are nothing more than small look-up tables that transform groups of 4 or 8 bits. These tables must be chosen very carefully to be as strong as possible against attacks, so they must be as nonlinear as possible and must not have statistical biases. To perform the permutation, however, it can be done simply by changing the order of the bits or by using basic linear algebra and matrix multiplication to mix their order [1].

The most widely used cipher in the world is one of these block ciphers and uses the substitution-permutation network as a technique. Its name is *Advanced Encryption Standard* (AES). During the Cold War, the United States and the Soviet Union created their own ciphers to secure their information. In the 1970s, the United States government developed the Data Encryption Standard (DES), which became a federal standard in 1979. In response, the KGB created GOST 28147-89 in the 1980s, which remained classified until the dissolution of the Soviet Union in 1990.

During the late 1990s, the security of DES was questioned, mainly because of the increase in available computing power that made brute-force attacks possible, but also because it was no longer efficient in terms of software implementations. In 1997, the *National Institute of Standards and Technology* (NIST) asked for proposals for the new Advanced Encryption Standard. The selection for this algorithm was an open process administered by NIST, where the advantages and disadvantages of the submitted ciphers were discussed in three rounds of evaluation. To submit an algorithm, the requirements were as follows:

- block cipher with 128-bit block size

- three key lengths must be supported: 128, 192 and 256 bit
- security relative to other submitted algorithms
- efficiency in software and hardware

On October 2, 2000, NIST announced their choice was the Rijndael algorithm, developed by two Belgian cryptographers, Rijmen and Daemen. Formal approval for inclusion in the US federal standard was made on November 26, 2001. Nowadays, the AES algorithm is used also in many commercial standards, including the Internet security standard IPsec, TLS, the Wi-Fi encryption standard IEEE 802.11i, the secure shell network protocol SSH (Secure Shell), the Internet phone Skype and numerous security products around the world [10].

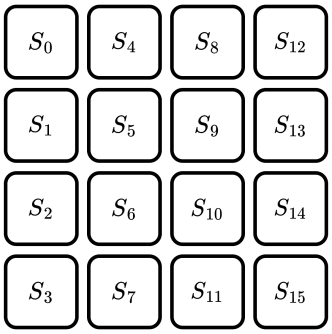
In this chapter, the architecture of the AES algorithm is be shown in section 2.2, paying particular attention to the building blocks that are part of it. In section 2.3 all the various modes of operation with which it can be performed is explained, while in section 2.4 is studied the main cryptanalyses that can be performed on AES and their feasibility.

2.2 Internal Structure

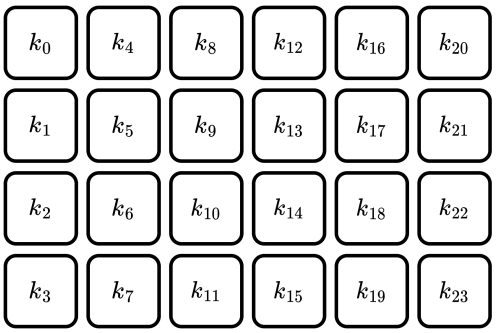
The AES algorithm is a *symmetric encryption block cipher* that processes 128-bit blocks using a secret key that can be 128, 192, or 256 bits long, although the 128-bit option is more commonly used because it is faster and the difference in security between 128 and 256 bits is minimal. For encryption (and decryption), AES sees the 16-byte plaintext as a 4x4 array, as in Figure 2.1(a), and then manipulates the individual bytes, rows, and columns of this array to arrive at the final result. The secret key is viewed in the same way, which, depending on whether it is made up of 128, 192, or 256 bits, is rearranged so that it forms a 4x4, 4x6, or 4x8 array respectively. As an example, in Figure 2.1(b) we can see the key matrix in the case of a 192-bit key.

To transform the plaintext, AES uses the Substitution-Permutation Network (SPN) structure which is shown in Figure 2.2. The four building blocks used by each round of the algorithm are *AddRoundKey*, *SubBytes*, *ShiftRows*, and *MixColumns*, and the number of rounds depends on the length of the key: a 128-bit key will have 10 rounds, a 192-bit key will have 12 rounds, and a 256-bit key will have 14 rounds. All rounds are identical except the last one, where the MixColumns operation is not performed. This is done because it saves unnecessary operations. To conclude the structure, an initial round is performed at the beginning composed of only the AddRoundKey.

Each round uses a different key that is generated by the *KeyExpansion* algorithm. This expansion, starting from the initial key, creates 11 round keys of 16 bytes in



((a)) The internal state of AES viewed as a 4x4 array of 16 bytes



((b)) The key matrix for a 192-bit key viewed as a 4x6 array of 24 bytes

Figure 2.1: Graphical representation of how the plaintext and key are manipulated within AES

the case of the 128-bit key. In the other two cases the length of the key always remains the same, what changes is the number of keys: 13 in the case of the 192-bit key, 15 in the case of 256-bit.

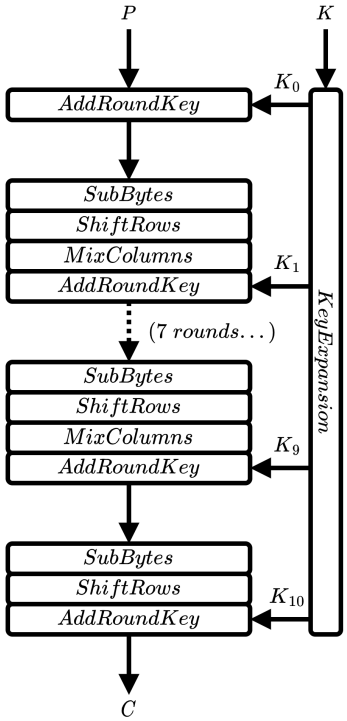


Figure 2.2: The internal operations of AES

These operations ensure the security of the AES algorithm. Each operation

contributes differently to it:

- *KeyExpansion* is used to generate a different key for each round, otherwise, if they were all the same, the cipher would be vulnerable to slide attacks;
- *AddRoundKey* is used to make encryption dependent on the key, otherwise anyone could decrypt the ciphertext without the key;
- *SubBytes* is used to add a non-linearity to the algorithm. Without it, it would just be a series of easily solvable algebraic operations;
- *ShiftRows* is used to make changes in a given column affect all other columns as well;
- *MixColumns* is used to make changes in one byte affect the other bytes in the state.

Most of the layers in AES, especially in the S-box and MixColumns layer, use finite field arithmetic, often called *Galois field*. It is a finite set of elements where addition, subtraction, multiplication, and division are defined. To understand better we need to introduce the concept of group. A group is a set with an operation and its corresponding inverse operation, for example if the operation is addition, its inverse operation is subtraction. In order to have all four elementary arithmetic operations in a single structure we need a set that contains an additive and multiplicative group. This is what is called a field.

In general, in cryptography it is always of interest to have a field with a finite number of elements, what is called a Galois field. The number of elements in the field is called the order or cardinality of the field, and is denoted by the expression $GF(p)$, where p is the number of elements. In AES the finite field contains 256 elements and is denoted as $GF(2^8)$. This field was chosen because each element can be represented by a byte. Therefore AES sees each byte of the internal data as an element of the field $GF(2^8)$, and manipulates this data by performing operations in this finite field [10].

2.2.1 SubBytes

The first transformation that is performed in each round is *SubBytes*. The operation that is performed is very simple: each byte of the state a_i is replaced by another byte b_i, like in Figure 2.3.

$$b_i = S(a_i) \tag{2.1}$$

To perform this substitution, 16 identical substitution-boxes (S-boxes) are used, each with 8 input and output bits.

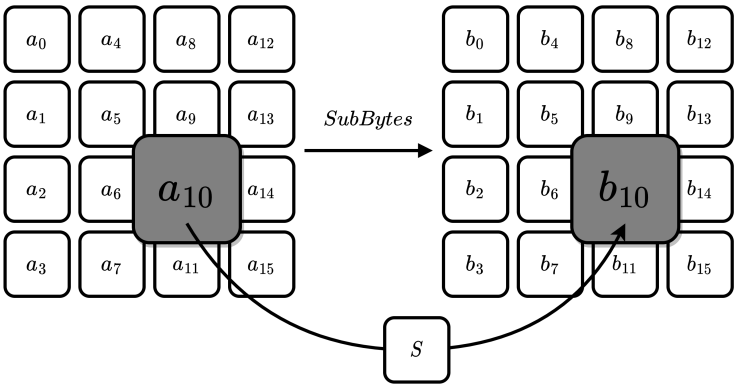


Figure 2.3: In the SubBytes step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table, S; $b_i = S(a_i)$.

In the AES algorithm, the S-box is the only nonlinear element. The substitution performed is a *bijective mapping*, which means that for every possible input there is one and only one output to which it is mapped, thus allowing to perform its inverse, which is necessary during decryption.

The AES S-box implementation has a strong algebraic structure, which can be viewed as a two-step mathematical transformation. The first part of the substitution is a *Galois field inversion*: for each input element A_i the inverse is computed as $B_i^{'} = A_i^{-1}$. In the second part of the transformation, each byte $B_i^{'}$ is multiplied with a constant bit-matrix followed by the addition with a constant 8-bit vector. This operation, called *affine mapping*, is described by the equation 2.2.

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0^{'} \\ b_1^{'} \\ b_2^{'} \\ b_3^{'} \\ b_4^{'} \\ b_5^{'} \\ b_6^{'} \\ b_7^{'} \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} mod 2 \tag{2.2}$$

Calculating these two steps for all 256 possible inputs would produce the Table 2.1, which represents the S-box of the Rijndael algorithm.

In most implementations, especially software ones, these two operations are not done explicitly but lookup tables are used instead. However, in hardware implementations, it is more advantageous to do the inversion followed by the affine mapping to realize the S-box.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Table 2.1: Rijndael S-box

2.2.2 ShiftRows

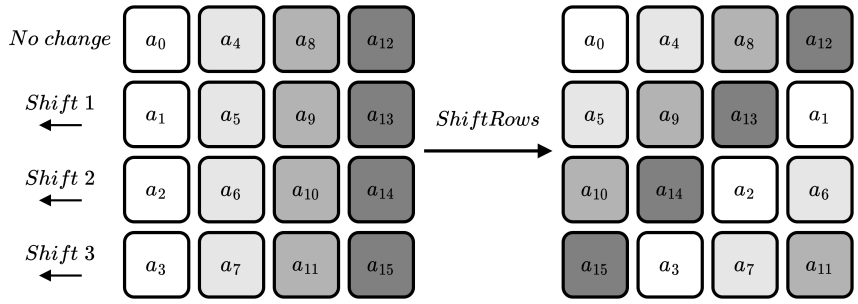


Figure 2.4: In the ShiftRows step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs incrementally for each row.

After the *SubBytes* layer, the ShiftRows operation follows in each round. This transformation cyclically shifts the second row one bytes to the left, the third row two bytes to the left, the fourth row three byte to the left, while leaving the first row untouched, as shown in Figure 2.4.

The main purpose of this transformation is to increase the diffusion properties of the algorithm by making each column of the output state consist of one byte of the column of the input state.

2.2.3 MixColumns

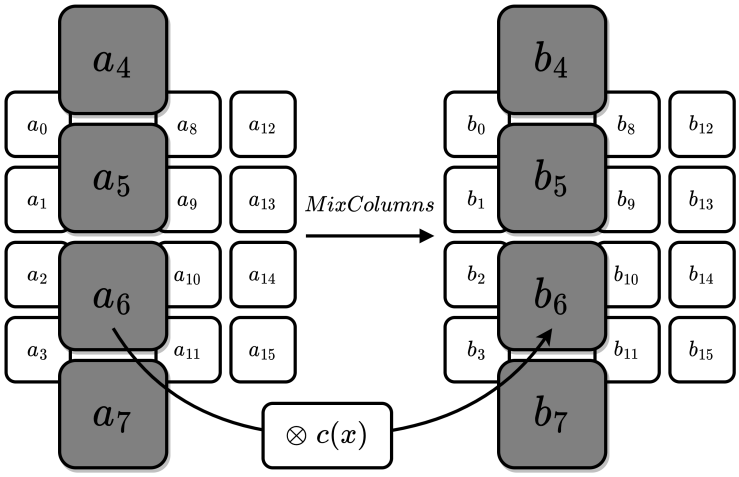


Figure 2.5: In the MixColumns step, each column of the state is multiplied with a fixed polynomial $c(x)$.

The *MixColumns* step is a linear transformation that mixes each column of the state matrix, making this operation the largest contributor to diffusion in AES as each input byte affects four output bytes. MixColumns together with ShiftRows ensure that only after three rounds each byte of the state matrix is dependent on all 16 input bytes of the plaintext.

To perform this transformation, each 4-byte column is considered a vector and multiplied by a 4x4 matrix containing constant elements, as shown in Figure 2.5. The operation that is performed for the first column is the one in equation 2.3.

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} A_0 \\ A_5 \\ A_{10} \\ A_{15} \end{pmatrix} \tag{2.3}$$

To perform this operation for the other columns the 4x4 matrix remains the same, only the corresponding column vector changes.

2.2.4 AddRoundKey

The last transformation that is performed is *AddRoundKey*, which is shown in Figure 2.6. The execution of this transformation is very simple. As input we have the 16 bytes of the state matrix and the 16 bytes of the round subkey, and they are combined together using the XOR operation. The value of the subkey is calculated, starting from the secret key, using Rijndael's key schedule.

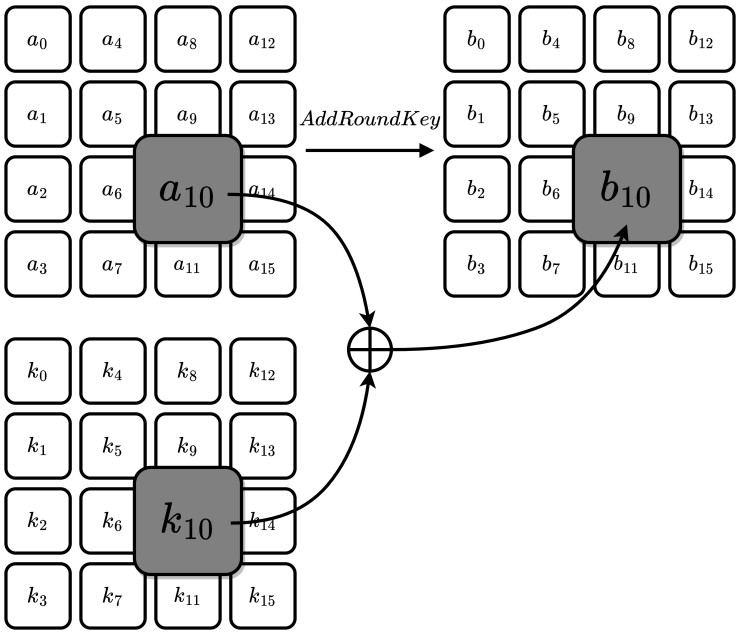


Figure 2.6: In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using the XOR operation ($\oplus$).

2.2.5 KeyExpansion

The number of keys needed to run the AES algorithm is equal to the number of rounds (remember that for the three different key lengths there are three different numbers of rounds, 10, 12, and 14 respectively) plus one, since a key is also needed for the initial XOR addition, also called key whitening. The mechanism for obtaining the subkeys is recursive, so to calculate the key K_{i+1} we need the key K, and how the subkeys are generated is different in the three cases, even if very similar.

The AES *key schedule* is based on *words*, which in this case are 4-byte words. For the key length of 128 bits we have four initial words that are formed by the 16 bytes of the secret key (the first word is made from the first 4 bytes, the second by the following 4 bytes and so on). KeyExpansion starts with taking the first word and calculates the first word of the first subkey by doing a XOR addition with the last word modified through a series of transformations. The first of these transformations is a simple left rotation of the bytes, and what is obtained is passed through an S-box (which is the same one used in the actual algorithm) to perform a byte-wise substitution. The last thing before doing the XOR is to add a *round coefficient* RC. This coefficient is an 8-bit coefficient that is added only to the leftmost byte, and it varies from round to round following the rule in equation 2.4 where i is the number of the iteration.

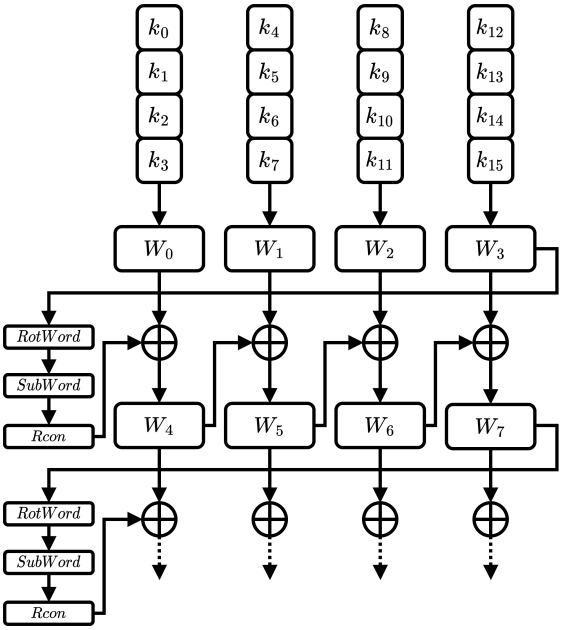


Figure 2.7: AES KeyExpansion for a 128-bit key.

$$rc_i = \begin{cases} 1 & \text{if } i = 1 \\ 2 \cdot rc_{i-1} & \text{if } i > 1 \text{ and } rc_{i-1} < 80_{16} \\ (2 \cdot rc_{i-1}) \oplus 11B_{16} & \text{if } i > 1 \text{ and } rc_{i-1} \geq 80_{16} \end{cases} \tag{2.4}$$

RC values for this key size are shown in Table 2.2.

i	1	2	3	4	5	6	7	8	9	10
rc_i	01	02	04	08	10	20	40	80	1B	36

Table 2.2: Values of rc_i in hexadecimal

The purpose of these transformations is to add nonlinearity to the KeyExpansion, but also to remove symmetry in AES.

The remaining three words are computed recursively by doing a XOR addition with the previously found word and the initial word. These operations are performed for ten iterations, and in this way the eleven necessary keys are finally obtained. The general structure of the KeyExpansion is shown in Figure 2.7.

For the 192-bit key, the KeyExpansion procedure is almost similar to that for the 128-bit key. The difference is that in this case there are eight iterations to calculate the 13 keys needed, and each iteration calculates six new words, arriving

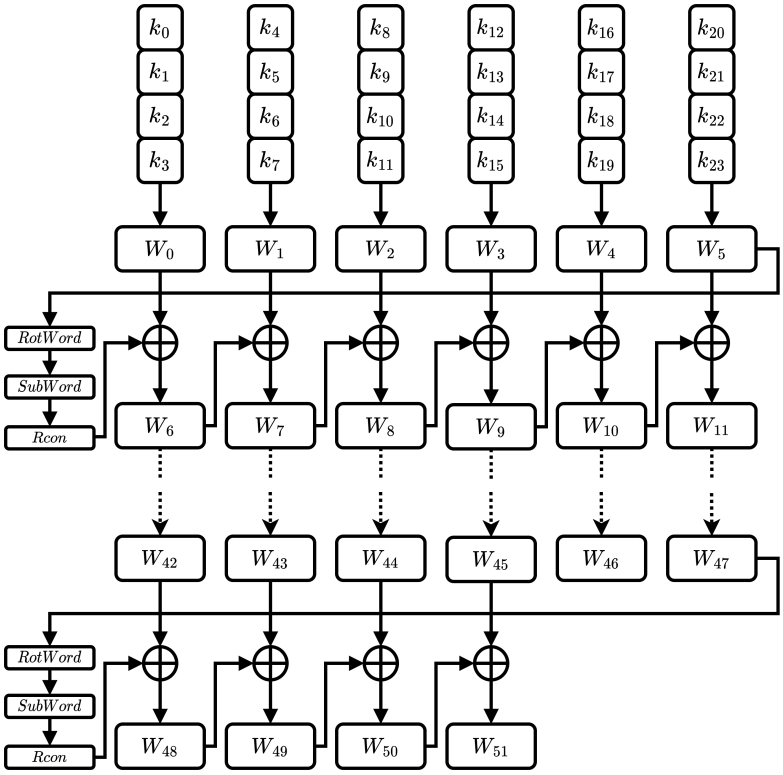


Figure 2.8: AES KeyExpansion for a 192-bit key.

at calculating up to fifty-two words (the last iteration calculates only four). Each subkey is made up of four words, so the one for the first round is made up of the words W_0, W_1, W_2 and W_3, the second one by W_4, W_5, W_6 and W_7, and so on. Since in this case we only have eight iterations, the RC coefficients that is needed in the transformation is only the first eight that are shown in Table 2.2. The general structure of the KeyExpansion for the 192-bit case is shown in Figure 2.8.

For the last key length, the one of 256 bits, the KeyExpansion structure is very similar to that for the 192-bit key. In this case we have seven iterations, and in each iteration eight words are produced, except in the last one which only four are generated. The main difference compared to the 192-bit case is that in each iteration to generate the fifth word we do not use the previous word directly as done previously, but this word is transformed before doing the XOR addition, as is done to generate the first word. In this case, however, only a simple byte-wise substitution is performed using the S-box. As for the 192-bit case, the subkeys are then formed by the words in order, so the first subkey is formed by the words W_0, W_1, W_2 and W_3 and so on. Since there are only seven iterations, the round coefficients will be only seven, and they are calculated as in the previous cases.

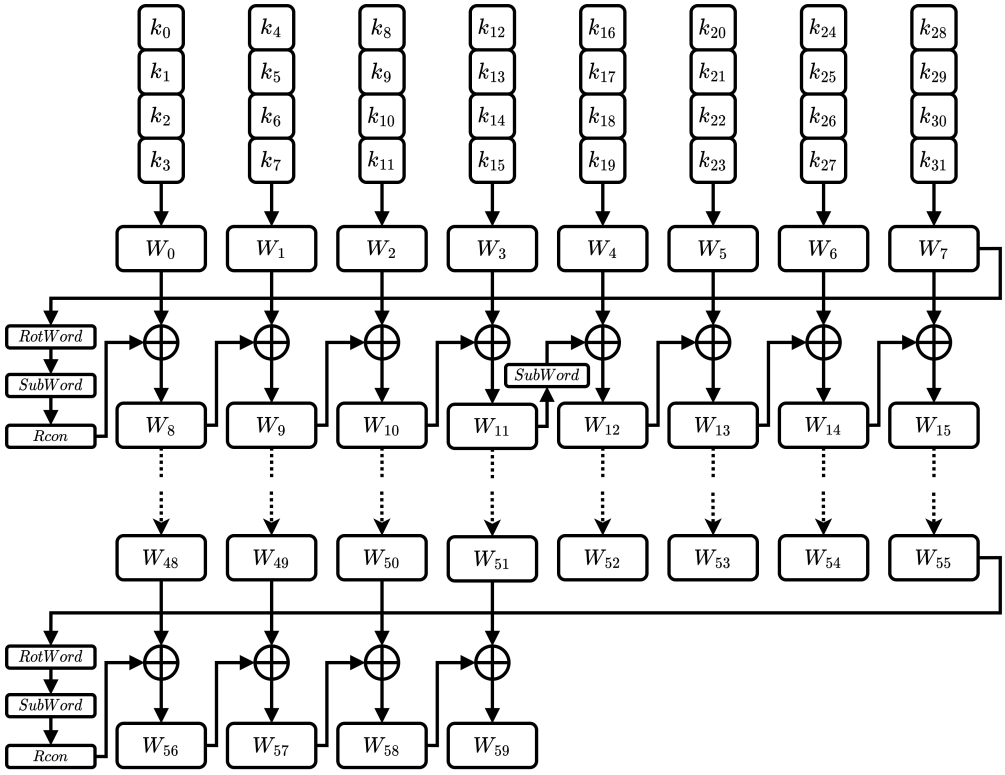


Figure 2.9: AES KeyExpansion for a 256-bit key.

Their values are the first seven in Table 2.2. Figure 2.9 represent the complete structure of the 256-bit KeyExpansion.

In the three cases, the generation of subkeys can be done in two ways, depending on the device that is used to perform the encryption or decryption. The first approach is called Precomputation in which all the subkeys are generated at the beginning of the cryptographic operation and then the encryption (or decryption) is performed. This approach is often used in devices that use the same key to encrypt large pieces of data, so it is preferable to use it in PCs or servers. However, it should be noted that in this case all the subkeys must be memorized, so it can only be done in devices that have a large memory available.

The second approach is called On-the-fly, where the subkey is generated for each round of encryption or decryption. This approach is used in smart cards, where the available memory is limited. Note in this case, however, that the decryption starts with the last subkey and then use them backwards, so this operation takes longer than the encryption due to this overhead.

2.2.6 Combining the AES layers

After explaining in detail how the various layers used in the AES algorithm work, this subsection shows how they are used in the block cipher to encrypt a plaintext using the pseudocode in Algorithm 1.

Algorithm 1 AES pseudocode

Require: p: 128-bit plaintext block
Require: k: key (128, 192, or 256 bits)
Ensure: c: 128-bit ciphertext block
▷ **Step 1: Key Expansion**
$roundKeys \leftarrow \text{KeyExpansion}(k)$
▷ **Step 2: Initial AddRoundKey**
$state \leftarrow p \oplus roundKeys[0]$
▷ **Step 3: Main Rounds (Nr-1 iterations)**
for $round \leftarrow 1$ to $Nr - 1$ **do** ▷ Nr = number of rounds (10, 12, or 14)
    $state \leftarrow \text{SubBytes}(state)$
    $state \leftarrow \text{ShiftRows}(state)$
    $state \leftarrow \text{MixColumns}(state)$
    $state \leftarrow state \oplus roundKeys[round]$ ▷ AddRoundKey
end for
▷ **Step 4: Final Round (without MixColumns)**
$state \leftarrow \text{SubBytes}(state)$
$state \leftarrow \text{ShiftRows}(state)$
$state \leftarrow state \oplus roundKeys[Nr]$
▷ **Resulting ciphertext**
$c \leftarrow state$

As can be seen, the only things the algorithm needs to perform the encryption are the 128-bit plaintext that will be manipulated, and the secret key, which can be 128 bits, 192 bits, or 256 bits depending on the version of AES used. The first operation that is performed is the generation of the round keys through the KeyExpansion layer. Some implementations may have a variant of this layer where the keys are generated one by one at the end of each round, but in this example they are all generated at the beginning in a single list. The next operation is the initial round composed only of the AddRoundKey layer, in which the plaintext is xored with the first generated key. Next is the main core of the algorithm, where round transformations, composed of the SubBytes, ShiftRows, MixColumns, and AddRoundKey layers, are iteratively applied to the state matrix. The number of rounds performed can vary depending on the AES version, 9 rounds for AES-128, 11 rounds for AES-192 and 13 rounds for AES-256, and at each round, one of the

keys generated previously at the beginning is used in the AddRoundKey layer. The last operation that is performed is the final round, which is the same as the previous rounds with the small difference that in this case the MixColumns layer is omitted. The state that is generated through these steps is then returned, and it is the ciphertext.

2.3 Modes of Operation

Block ciphers, including AES, can use different *modes of operation* to encrypt plaintext. Below we will analyze some of them, starting from the simplest, but also the least secure, called electronic codebook.

2.3.1 Electronic Codebook (ECB) Mode

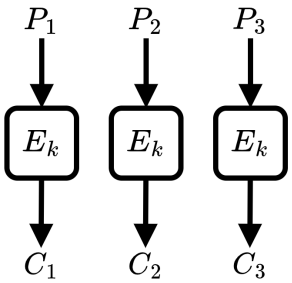


Figure 2.10: The ECB mode.

With *ECB* mode, plaintext blocks are taken and processed independently by computing $C_1 = E(K, P_1)$, as can be seen in Figure 2.10. With this mode, identical ciphertext blocks show identical plaintext blocks to an attacker, making it unsafe. Another problem with ECB is that it only takes complete blocks of data, so in the case of AES, which processes 16-byte blocks, the encryption takes pieces of data that are a multiple of 16 bytes in size, although there are ways to overcome this.

2.3.2 Cipher Block Chaining (CBC) Mode

Cipher Block Chaining (CBC) mode is very similar to ECB mode, with the difference that all ciphertexts are chained together. In fact, the plaintext is not encrypted as it is, but first a XOR is made with the ciphertext obtained from the previous block, as shown in Figure 2.11. So in this case the operation that is performed is $C_i = E(K, P_i \oplus C_{i-1})$. In the case of the first block, in which there is no previous ciphertext, the XOR operation is done with a random *initial value* (IV), which

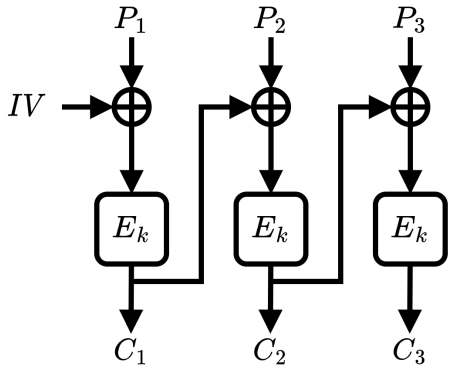


Figure 2.11: The CBC mode.

must be different for each encryption operation. The IV will then be sent together with the ciphertext because it is needed to perform the decryption. So CBC makes each ciphertext block dependent on the previous blocks, and in this way identical plaintexts are encrypted into different ciphertexts.

In this mode, decryption can be much faster than encryption, since to do the latter it is necessary to know the value of the previous ciphertext, and therefore it is necessary to wait for the encryption of the block that comes before to finish. In the case of decryption, however, the operation that is performed is $P_i = D(K, C_i) \oplus (C_{i-1})$, and therefore, already knowing the value of the entire ciphertext, it is possible to perform it in parallel simultaneously for all blocks, saving time.

In case the plaintext is not a multiple of the block length (16 bytes for AES) there are some techniques to solve this problem. The first technique is called *padding*, and it is used to expand the plaintext to reach the block length by adding extra bytes following the following rule: add as many bytes as necessary all with the value of the number of bytes added.

Example. For example if the plaintext is 1 byte long, add 15 bytes 0f (15 in hexadecimal), or if the plaintext is 11 bytes long, add 5 bytes 05 (5 in hexadecimal). In case the plaintext is already 16 bytes, 16 bytes 10 (16 in hexadecimal) must be added.

The decryption in this case is identical only with one additional step: after decrypting the ciphertext, it is checked that at the end there are as many bytes at the bottom as indicated by the last byte. If this is not satisfied, the message is

rejected, otherwise the padding bytes are eliminated and the remaining bytes are returned.

Another trick to be able to encrypt with blocks that are not of the desired length is what is called *ciphertext stealing*. This method is more complicated and less used than padding, but it has several advantages. The first is that the plaintext can be of any bit length and the ciphertext is exactly the same length as the plaintext. In fact, one of the drawbacks of padding is that it makes the ciphertext longer by at least one byte and at most one block. The second advantage that can be obtained with ciphertext stealing is that it is not vulnerable to padding oracle attacks, which are very powerful attacks that can sometimes work against the CBC with padding.

The operation is as follows. Encryption occurs unchanged for all blocks except the last and the penultimate block. The penultimate block is encrypted, and its output is truncated to the length of the last block. The truncated portion is then added to the last plaintext, and encryption is then performed. Finally, the resulting ciphertext in the last block is returned as the ciphertext of the penultimate block, while the truncated ciphertext of the penultimate block is returned as the ciphertext of the last block. Decryption is simply the reverse operation.

2.3.3 Output Feedback (OFB) Mode

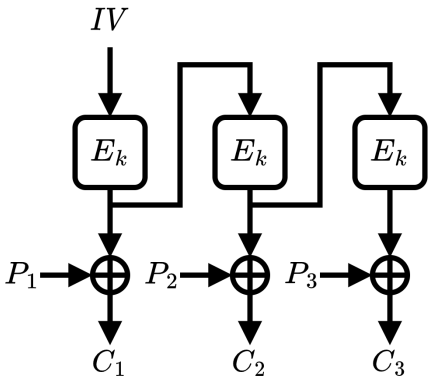


Figure 2.12: The OFB mode.

Output feedback (CFB) mode is different from ECB and CBC in that it turns a block cipher into a *stream cipher*. The idea behind CFB mode is very simple, and is shown in Figure 2.12. AES is used to encrypt an IV, thus creating the first stream key. This output is then encrypted in the next block to generate the second stream key, and so on. These stream keys are then XORed with the plaintext to obtain the corresponding ciphertext, one stream key for each block of plaintext. Since this

is a stream cipher, encryption and decryption are exactly the same operation, so AES is always used in encryption mode.

Since an IV is used, the output of encryption using OFB is non-deterministic, since encrypting the same message twice results in two different ciphertexts. As in the CBC case, since the IV is different for each message, it must be passed along with the ciphertext in order to decrypt.

An advantage of the OFB mode is that the encryption procedure can start before receiving the plaintext, because to calculate the stream keys it only need the IV and the previous key. Once calculated, the blocks of the plaintext can be xored even in parallel.

2.3.4 Cipher Feedback (CFB) Mode

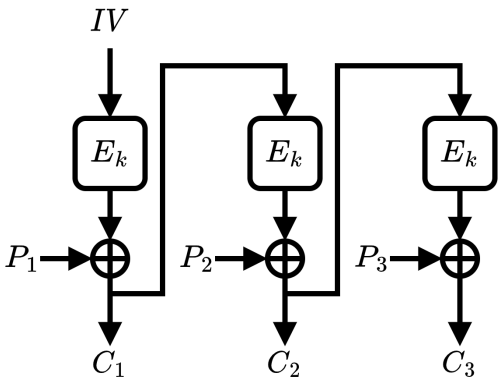


Figure 2.13: The CFB mode.

Cipher feedback (CFB) mode is not very different from OFB mode, as can be seen in figure 2.12. The only difference is that instead of using the block cipher output, the ciphertext is used to calculate the stream key of the next block. So everything that has been said for OFB also applies to this mode. However, using the ciphertext to calculate the stream key creates an asynchronous stream cipher, and so in this case the advantage of precalculating the values of the stream keys in order to be able to XOR with the plaintext in parallel at a later time is lost.

2.3.5 Counter (CTR) Mode

Counter mode (CTR) is very useful to have the same benefits as ciphertext stealing, although it is not a block cipher mode because it turns even in this case a block cipher into a *stream cipher*. How this works can be seen in figure 2.14. The plaintext is not encrypted using AES, but instead it is XORed with an encrypted

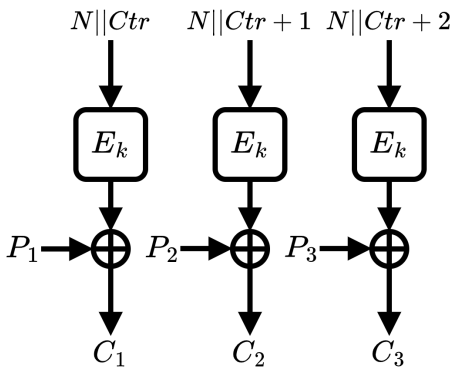


Figure 2.14: The CTR mode.

block consisting of a counter, ctr, and a nonce, N. A counter is an integer that is different for each block, so no two blocks can have the same counter, but it is possible to use the same sequence of counters for different messages. A nonce is a number that is used only once. All blocks have the same nonce, but all messages must have a different nonce, so it is provided by the encryption and is sent together with the ciphertext. Decryption is the same, requiring the encryption algorithm to do both encryption and decryption.

The main benefit of CTR mode is that it can be the fastest compared to other modes, since not only can it be run in parallel, but it can start the encryption procedure when the plaintext is not yet provided by taking a nonce and calculating the stream that is then be XORed with the plaintext later.

2.4 Cryptanalysis

The AES algorithm, as mentioned above, is one of the most widely used cryptographic algorithms in the world, and is famous for its security and robustness. However, this does not mean that it is not completely immune to attacks. In the following sections are investigated some of the possible attacks that can be used to break AES and study their actual feasibility in doing so.

2.4.1 Brute-Force Cryptanalysis

Brute-force, or Exhaustive Key Search, is the simplest attack that can be performed and is based on a simple concept: the attacker knows the ciphertext and the plaintext, and by trying all the possible values that the key can assume to decrypt the ciphertext, if the plaintext that is obtained is equal to the original plaintext then the key used to perform the decryption is the correct one. In theory, these

types of attacks are always possible against symmetric ciphers, such as AES, but their feasibility depends greatly on the key space, that is the number of possible keys for a given cipher. If testing all of them requires an unsustainable amount of time, then the cipher can be considered secure against brute-force attacks [10].

The AES algorithm has three possible key lengths: 128, 192, and 256 bits. In these three cases, the possible combinations that can be obtained are 2^{128}, 2^{192}, and 2^{256}, respectively. Let's take the weakest case, that is the key length 128 bits, and suppose we have a computer capable of testing a billion keys per second, that is 10^9. To be able to perform a brute-force attack under these conditions, it would take $\frac{2^{128}}{10^9} \simeq 3.4 \cdot 10^{29}$ seconds, which equals $1.08 \cdot 10^{22}$ years. For comparison, the age of the universe has been estimated to be 13.787 billion years, or $1.3787 \cdot 10^{10}$. This makes it clear how impractical an attack of this kind is.

But the infeasibility does not end there since not only would it require an astronomical amount of time, but also the energy that would be spent to execute the attack would be disproportionate, and the memory necessary to be able to run such a large amount of calculations in parallel is not technologically possible.

2.4.2 Biclique Cryptanalysis

In 2011, the research group consisting of Andrey Bogdanov, Dmitry Khovratovich and Christian Rechberger proposed in [11] an advanced technique to optimize the brute-force attack called *biclique*. This attack is based on the idea of dividing the AES algorithm into two partial phases, the first part concerning the first rounds of AES, the second instead concerning the final rounds. After that, the intermediate result of the initial rounds is calculated for a subset of candidate keys, and the same thing is done backwards starting from the ciphertexts and passing through the final rounds. Finally, a grid is created in which in each row are placed the intermediate results derived from the first part of the encryption, and in each column are placed the results of the decryption of the second part. In this way the intersection between the two results is facilitated, allowing the test to be done in parallel and therefore reducing the total number of operations to be performed.

Although the number of operations needed to test a key is reduced, the attack is still impractical in reality since the attack complexity in the three cases goes from 2^{128} to $2^{126.18}$ for the 128-bit key, from 2^{192} to $2^{189.74}$ for the 192-bit key, and from 2^{256} to $2^{254.42}$ for the 256-bit key. So even in this case the time needed to execute this improved brute-force attack remains much greater than the age of the universe, and consequently the energy expended and the memory required also remain too expensive for current technology.

2.4.3 Linear Cryptanalysis

Linear cryptanalysis is one of the most important general techniques for analyzing a block cipher. Although it is less successful than differential cryptanalysis, which will be explained later, it has the advantage that cryptanalysis only needs the plaintext to be able to be done. Linear cryptanalysis aims to find an approximate linear relationship between the plaintext bits, the ciphertext bits, and the secret key bits. This technique was designed by Mitsuru Matsui in [12][13] for analyzing the DES cipher, but can also be applied to other block ciphers including AES.

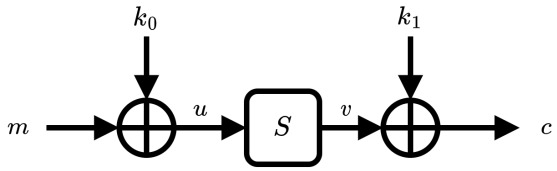


Figure 2.15: Simple encryption system used as an example to explain linear cryptanalysis

The attack works as follows. Let's take a simple encryption system as an example, the one that is showed in Figure 2.15. To compute the ciphertext with it the following operations are performed: $u = m \oplus k_0$; $v = S[u]$; $c = v \oplus k_1$. We have that m is the message, k_0 and k_1 are two randomly chosen four-bit round keys, $S[\cdot]$ is a four-bit permutation, and c is the ciphertext. The input, output and key blocks are seen as column vectors of bits, so to identify a specific bit it is required to premultiply the column vector by a row vector that works as a mask. For example if we want to identify a single bit in the message, the mask will the one in equation 2.5.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} m_3 \\ m_2 \\ m_1 \\ m_0 \end{pmatrix} = m_3 \tag{2.5}$$

In order to approximate the permutation $S[\cdot]$ we assume that we can find a pair of masks (α, β) such that $(\alpha \cdot x) = (\beta \cdot S[x])$ with probability $p \neq \frac{1}{2}$. So considering all the operations of the cipher we have $(\alpha \cdot m) = (\alpha \cdot k_0) \oplus (\alpha \cdot u)$ with probability 1, $(\alpha \cdot u) = (\beta \cdot v)$ with probability p, and $(\beta \cdot v) = (\beta \cdot k_1) \oplus (\beta \cdot c)$. Adding these equations and rearranging the terms we get the equation 2.6.

$$(\alpha \cdot m) \oplus (\beta \cdot c) = (\alpha \cdot k_0) \oplus (\beta \cdot k_1) \text{ with probability p} \tag{2.6}$$

Note that in this equation on the left we have an expression that depends only on the bits of the message and the ciphertext, while on the right we have an expression that depends only on the bits of the two round keys k_0 and k_1. If the probability of this equation is equal to 1, then by calculating the argument on the left we can deduce one bit of information about the keys. The same thing happens if the probability is 0 because by modifying the equation in 2.7

$$(\alpha \cdot m) \oplus (\beta \cdot c) \oplus 1 = (\alpha \cdot k_0) \oplus (\beta \cdot k_1) \tag{2.7}$$

we have that the probability becomes 1, and therefore also in this case we can deduce one bit of information about the keys. These two situations are ideal for a cryptanalyst. The worst case is when the probability is equal to $\frac{1}{2}$, since the left-hand side equals the right-hand side half the time, and so when averaging all possible inputs it gives no information about the bits of the keys. So the goal of linear cryptanalysis is to choose masks α and β such that the equations in the linear approximation have probability $p = \frac{1}{2} + \epsilon$, with $|\epsilon|$, called bias, as large as possible.

Let's take as an example the case where $\alpha \cdot x = \beta \cdot S[x]$ has two matches, and therefore its probability is $\frac{2}{16}$. Then the equation 2.7 has probability $p = \frac{14}{16}$ and with this we can understand the basis of the attack. Using two counters T_0 and T_1 initialized to 0, N encryptions of N known plaintexts are made. For each plaintext-ciphertext pair, the argument on the right side of the equation is estimated by calculating the argument on the left side of the equation, and if the result is 0, the counter T_0 is incremented, otherwise if the result is 1, the counter T_1 is incremented. We know that the key bit estimate has a probability of $\frac{14}{16}$, and so after processing the N plaintexts the counter T_0 has the value $\frac{2N}{16}$ and the counter T_1 has have the value $\frac{14N}{16}$ if $(\alpha \cdot k_0) \oplus (\beta \cdot k_1) = 1$, otherwise if $(\alpha \cdot k_0) \oplus (\beta \cdot k_1) = 0$ then T_0 has the value $\frac{14N}{16}$ and T_1 the value $\frac{2N}{16}$. So by looking at the value of T_0, the value of a bit of the key can be determined. To increase the probability of success with this attack, it is necessary to increase the number N of plaintexts used [14]. We can see a graphical representation of what has just been explained in figure 2.16.

A very useful metric that characterizes the resistance of a certain implementation against this type of cryptanalysis is *nonlinearity*, and it is calculated on S-boxes. The nonlinearity is defined as the minimum Hamming distance between the output function of an S-box and all *affine functions*, i.e. a function that can be expressed as in equation 2.8, where x is the n-bit input vector, A is a binary matrix of size $m \times n$, and b is a constant vector of m bits.

$$f(x) = Ax \oplus b \tag{2.8}$$

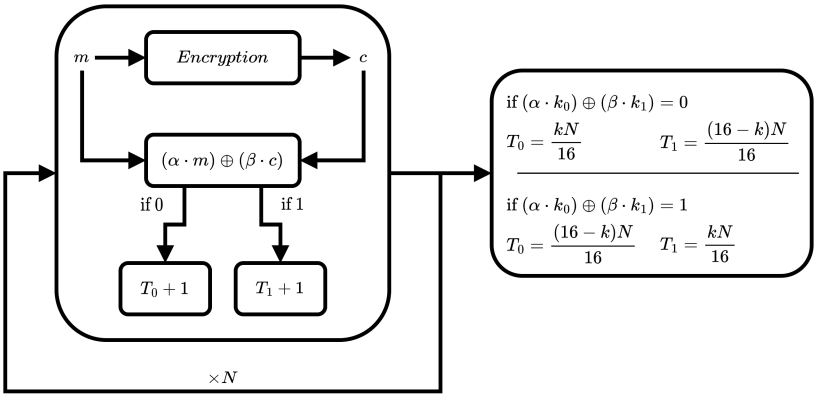


Figure 2.16: Graphical representation of linear cryptanalysis

An S-box must be as different as possible from these special functions, because affine functions are predictable and easy to describe mathematically, and linear cryptanalysis is based on this.

The higher the nonlinearity of an S-box, more robust it is against linear cryptanalysis. It has been studied that the lower this metric is, the more information is leaked. This metric is defined as in equation 2.9, where F is the S-box function and W_F is the Walsh transform of F.

$$NL(F) = 2^{n-1} - \frac{1}{2} max_{u \in \mathbb{F}_2^n; v \in F_2^{m*}} \left(|W_F(u,v)| \right) \tag{2.9}$$

In the case of (n, n) functions, for example S-boxes, the maximum that can be obtained for this property is described by the equation 2.10 [15].

$$NL \leq 2^{n-1} - 2^{\frac{n-1}{2}} \tag{2.10}$$

The closer this metric for an S-box is to this value, the better its resistance against linear attacks. In the case of AES, which uses an (8,8) S-box, the maximum value that can be obtained is 116.686. In the case of an (5,5) S-box, which is used by ASCON, presented in Chapter 3, the maximum value is 12.

As AES creators Joan Daemen and Vincent Rijmen explain in [16], AES was designed with the intention of being immune to linear cryptanalysis. In fact, this algorithm was designed following the "wide trail strategy", which effectively combines two fundamental principles of cryptography, diffusion and confusion. Diffusion is achieved by the MixColumns and ShiftRows layers, where the effect of a single bit is propagated throughout the entire block. Confusion is achieved by the S-box, which is designed to be highly nonlinear and resistant to linear approximations, making it difficult to find a simple relationship between input and output.

The number of rounds also plays a key role in defending against linear cryptanalysis. AES, using 10, 12, or 14 rounds, means that even if an attacker were to find a valid linear approximation for a single round, as more and more rounds are added, further nonlinearity and bit diffusion are added, making this linear approximation increasingly weaker.

Due to this design of the S-box and diffusion, the residual bias of a possible linear approximation becomes extremely small, thus making an extremely large data collection necessary to make this attack successful.

2.4.4 Differential Cryptanalysis

One of the most powerful attacks that can be launched against block ciphers is *differential cryptanalysis*. It was invented by Eli Biham and Adi Shamir in [17], and this technique aims to find predetermined differences between pairs of plaintexts and study how they propagate through the cipher to obtain information about the secret key.

To better understand how it works, let's take a simple cipher like the one considered in the case of linear cryptanalysis and shown in figure 2.15. The operations that are then performed are $u = m \oplus k_0$, $v = S[u]$ and $c = v \oplus k_1$. To calculate the differential behavior of this cipher, two plaintexts, m_0 and m_1, must be used, and again, the two corresponding ciphertexts c_0 and c_1 are calculated by performing the following operations: for plaintext m_0, $u_0 = m_0 \oplus k_0$, $v_0 = S[u_0]$ and $c_0 = v_0 \oplus k_1$, for plaintext m_1, $u_1 = m_1 \oplus k_0$, $v_1 = S[u_1]$ and $c_1 = v_1 \oplus k_1$. In order to know the values of u_0 and u_1, it is necessary to know the value of k_0 which is unknown. However, knowing the value of the two plaintexts, an attacker knows the value of the difference of these two internal values thanks to the relation in equation 2.11.

$$u_0 \oplus u_1 = (m_0 \oplus k_0) \oplus (m_1 \oplus k_1) = m_0 \oplus m_1 \tag{2.11}$$

This value can be used to find the secret key k_1, in fact a cryptanalyst can calculate the value of v_0 and v_1 using the ciphertexts and by making an assumption about the value that k_1 can assume. Once these values are found, since the S-boxes are public and invertible, the difference between u_0 and u_1 can be calculated. The operation that is performed is shown in equation 2.12.

$$u_0 \oplus u_1 = S^{-1}[t \oplus c_0] \oplus S^{-1}[t \oplus c_1] \tag{2.12}$$

Since this difference is known, by trying all the possible keys, those in which the expected value is equal to the real value are saved as candidate keys. A graphical

representation of what has just been explained is shown in figure 2.17 to better understand the operations that are performed. If more than one candidate key is found for k_1, the attack is repeated with other plaintext-ciphertext pairs until a unique solution is found.

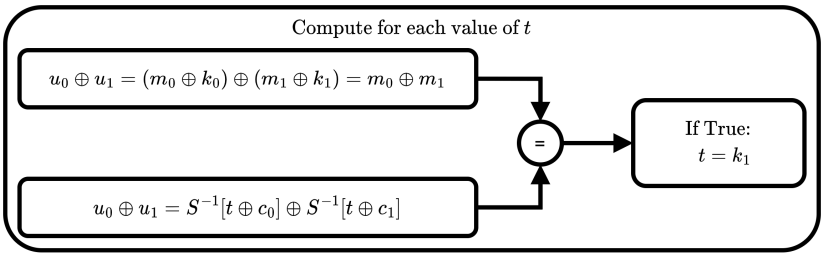


Figure 2.17: Graphical representation of differential cryptanalysis

This simple example highlights two important points about differential cryptanalysis. The first is that, even if the values of the internal variables are unknown, it is possible to determine the differences between them at certain points during the cryptographic operation. The second is that it is possible to derive information about the key by making assumptions about the secret key and seeing whether a certain differential condition is satisfied. This is the most common method of obtaining information about keys using differential cryptanalysis [14].

The metric that measures the resistance of an S-box against differential attacks is *differential uniformity*, which calculates the maximum probability that a difference in plaintext translates into a difference in ciphertext across the S-box. S-boxes that has small values of differential uniformity has the capability to defy differential cryptanalytic efforts. It can be calculated as in equation 2.13, where Δg is the input differential, Δf is the output differential, and K is the guessed key.

$$DU(F) = max_{\Delta g \neq 0, \Delta f}(\#\{g \in K | F(g) \oplus F(g \oplus \Delta g) = \Delta f\}) \tag{2.13}$$

Functions that have a differential uniformity of 2 are called *Almost Perfect Linear* (APN) functions. This value is the best that can be obtained for (n, n) functions where n is an odd value, or equal to 6, so for ASCON where $n = 5$. For values of n even and greater than 6, as in the case of AES, it is still an open question [15]. It has been found that a differential uniformity of 4, common for ciphers such as AES, is a good compromise between resistance to attacks and practical implementation.

As with linear cryptanalysis, [16] also explains the robustness against differential cryptanalysis. The reasons for this good design are due to the same elements that favor a good defense against linear cryptanalysis. In fact, the S-box has been designed in such a way as to minimize the probability of differential transitions,

and therefore making it more difficult to find such differences. As before, diffusion helps a lot also in this case since AES has a rapid diffusion through the rounds, causing any small difference in the input to be propagated very quickly throughout the block and therefore affecting all the bits in a few rounds. Finally, the number of rounds is such that it is impossible to trace and exploit the initial input differences after a full number of rounds. So, while this technique is very powerful against block ciphers, the way the AES algorithm was designed gives it unique properties that are useful against all attacks that have been analyzed so far.

2.4.5 Related-Keys Cryptanalysis

Related-key cryptanalysis was introduced by Eli Biham in [18] and is a technique that relies on the fact that in many block ciphers, the key scheduling algorithm can be viewed as a set of algorithms that extract a subkey from subkeys from previous rounds. If all of these algorithms are the same, then, given a key, one can shift all the keys from a round backwards to obtain a new set of valid subkeys that can be derived from another key. Hence the name related-key.

There are several variations of this attack, some of which allow the cryptanalyst to choose the relationship between the keys, while others only require knowledge of the relationship between them. Since related-key cryptanalysis relies heavily on the form of key scheduling in ciphers, and since it is often tailored to a single block cipher, it is not possible to provide a general recipe for this type of attack. However, to better understand how it works, an example of a related-key attack on Triple-DES will now be described.

For this attack we have two encryptions of the same message using two different keys $k = (k_1, k_2, k_3)$ and $k' = (k_1, k_2, k_3 \oplus s)$, where s is a non-zero value. Decrypting using single DES with all possible key values l yields that $DEC_l(c) = DEC_{l \oplus s}(c')$ when $l = k_3$. For any other value of l this equivalence is not valid, so one can find the value of k_3 by doing a brute-force attack that requires only 2^{56} operations. Obviously the value of s must be known otherwise this cryptanalysis would be much more complicated [14].

In [16] Joan Daemen and Vincent Rijmen states that "The key schedule of Rijndael, with its high diffusion and non-linearity, makes it difficult to mount attacks of this type". However in 2000, before Rijndael was chosen as the AES, Ferguson et al. in [19] disagrees with this statement. Indeed, Ferguson and his team discovered significant vulnerabilities in Rijndael's KeyExpansion, the most problematic of which is the slow diffusion of differences. This means that it can take several key schedule cycles for a small difference in one byte to propagate, allowing an attacker to easily track how an initial difference in a related key spreads across round keys, making it possible to identify predictable patterns.

Exploiting this vulnerability, Ferguson describes a related-key attack that can

be launched in which 256 related keys are generated that differ only by a specific byte of the fourth round key. The attack uses a series of plaintexts that after the first round of encryption produce an identical state despite the related keys, which allows for a difference to be introduced at the end of round 3 that propagates in such a way that every byte after the MixColumns operation of the fifth round takes on all possible values. Taking advantage of this situation, the attacker backcalculates the output of round 6 from the ciphertext, and calculates the sum of some specific bytes of it, if the result is equal to zero then it means that the attack was successful.

This attack can break AES-256 up to nine rounds, and by guessing 31 bytes of the key material, for each attempt the attacker must perform work equivalent to a single encryption, making the total complexity of the attack 2^{248}. By further improving the attack, the complexity is able to drop to $5 \cdot 2^{224}$, which is a significant improvement over a brute-force attack.

More recently, in 2009, Alex Biryukov and Dmitry Khovratovich in [20] managed to perform related-key cryptanalysis on the full AES-192 and AES-256 with complexity $2^{99.5}$ for data and time, and using 2^{77} words of memory for the latter. The complexity for AES-192 is highest with data complexity 2^{123}, time complexity 2^{176}, and using 2^{152} memory words.

While these attacks are feasible on paper, the complexity required to perform them is still too high. Taking the world's fastest computer, which can perform a trillion operations per second, to perform the related-key cryptanalysis designed by Biryukov and Khovratovich it would take 28,000 years, too long to be even remotely sustainable as an attack.

2.4.6 Side-Channel Cryptanalysis

As we have seen, the AES algorithm has been designed to be very efficient against all known types of attack, making it one of the most secure ciphers in the world. However, there is another type of attack that relies not only on the mathematical properties of the cipher, but also on the physical properties of its implementation: we are talking about *side-channel attacks*. The types of attacks of this type and how they work are described in detail in chapter 4, so it is not covered in this section. What interests us now is how secure the AES algorithm is against these types of attacks.

Starting from timing attacks, the only possible vulnerability that can be found in Rijndael is the implementation of finite field multiplication in MixColumns. All other operations in the algorithm are implemented by instructions that all take a constant time, so they are not a concern for this type of side-channel attack. The MixColumns transformation instead depends on mathematical operations that can have a variable execution time depending on the data they operate on, thus creating a weakness in the computation by inferring information about the intermediate

data. Furthermore, MixColumns is a linear transformation that takes the current state as input and produces an output that becomes the input of the next phase. If an attacker is able to observe the state before and after this operation, he could correlate this data and make assumptions about the content of the previous state or the key used.

Fortunately, as explained in [16], this vulnerability can be eliminated by using a look-up table to implement this operation, and then the execution time to perform the finite field multiplication becomes constant whatever the data to be transformed.

AES can equally easily defend itself against SPA attacks, since Rijndael can be implemented without difficulty with a fixed sequence of instructions, which makes it pointless to try to perform side-channel attacks of this type.

However, the implementations to protect against DPA attacks are more difficult than timing and SPA ones, especially with a high signal-to-noise ratio. The countermeasures proposed in [16] are different. The first one is formed by techniques to individually protect each instruction from power analysis. An example is load balancing, which is a methodology in which by redesigning the hardware one gains a minimization or complete elimination of the correlation between dissipated power and the operands used. This can also be simulated via software by making sure that each data word always contains both data bits and complemented data bits. Another technique is called masking of operands, in which the operand is replaced with two operands that are unpredictable to the attacker, and only the joint knowledge of both can reveal information about the original operand.

Another countermeasure instead tries to limit the impact of the vulnerabilities of individual instructions. To do this, a desynchronization can be done, that is, changing the order of the sequence of instructions for each encryption, and in this way making it more difficult for an attacker to collect statistics.

Special mention should be made of a weakness of AES that is widely exploited in DPA and CPA attacks, the *S-box*. AES S-boxes are designed to be highly effective against traditional cryptographic attacks, such as linear and differential analysis, due to their ability to introduce nonlinearity and confusion into the encryption process. However, this same operational complexity becomes a significant weakness when considering side-channel attacks, such as DPA and CPA. In these contexts, the nonlinearity of the S-boxes causes variations in the power consumption of the device that can be measured and analyzed to trace back to sensitive information, such as intermediate bits of the secret key. The strong correlation between S-boxes output and power consumption makes it possible for attackers to exploit these nonlinear operations to successfully perform SCA attacks, highlighting a major weakness of the AES system that, although robust against linear and differential analysis, can be compromised through techniques that analyze side channels. Therefore, it is clear that specific countermeasures need to be implemented

to mitigate these vulnerabilities inherent to S-boxes in AES. In chapter 5, one of these countermeasures is examined, which, with a slight modification of the implementation, attempts to reduce this vulnerability and make this algorithm more robust.

Chapter 3

ASCON

3.1 Introduction

In the cryptography landscape, encryption algorithms can be of many types. Block ciphers, of which AES is part, are a typology, but in some contexts there is a need not only to protect the confidentiality of a message, but also its authenticity. For this purpose, *hash functions* are used. They take a large message of any length as input and produce a shorter one of fixed length as output, called *hash*. If this operation is secure, two different messages that are manipulated using the same hash function will have two different hashes. This property can be exploited to ensure that the data has not been modified during the transfer, thus generating a real digital signature.

The vulnerability of this operation is that anyone can verify that a message has a certain message hash value because no secret value is involved in the operation. For this reason, the possibility of using a secret key to perform this operation has been added, thus realizing the so-called *keyed hashing*. This transformation is the basis of two very important cryptographic algorithms, *message authenticated codes* (MACs), which authenticate a message and protect its integrity, and *pseudorandom functions* (PRFs), which produce output values that appear random.

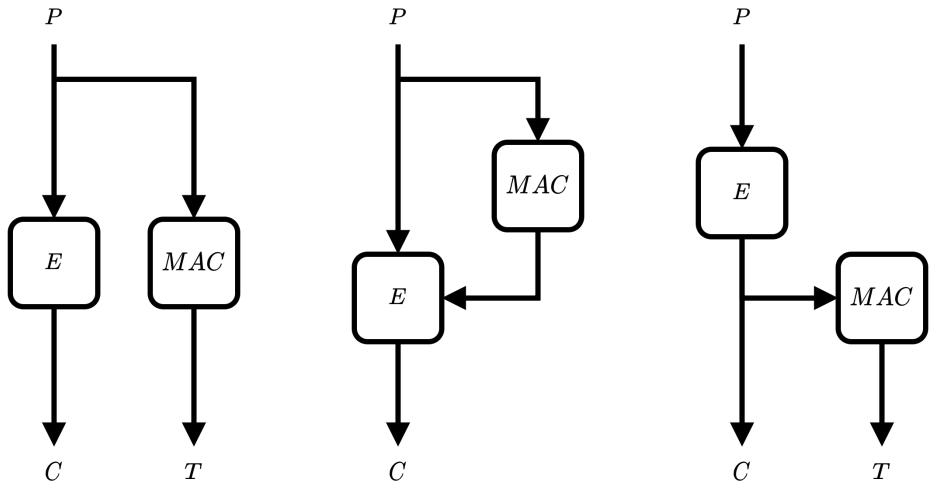
A MAC produces a so-called *authentication tag* T of a message M using a key K ($T = MAC(K, M)$), and in this way only those who know the key can authenticate that the message sent has not been altered after it has been sent. To have a more secure communication system, the MAC is often supported by a cipher, in order to also protect the confidentiality of a message. This system is used for example in the Internet Protocol Security (IPSec), Secure Shell (SSH) and Transport Layer Security (TLS) protocols, which generate an authentication tag for each packet they transmit.

Similarly a PRF uses a secret key to generate a seemingly random output

($PRF(K, M)$), but unlike the MAC, PRFs are not intended to be used alone but as an integral part of cryptographic algorithms or protocols. To be secure, a PRF must not have any patterns that cause its output to have a non-random shape, so that an attacker cannot tell whether he is talking to a PRF algorithm or a pseudorandom function. This condition makes a PRF stronger than a MAC, especially since a MAC is considered secure only if its output cannot be guessed, while a PRF is secure if its output cannot be distinguished from random strings, which is a stronger requirement.

Going back to the beginning of this introduction, there is a type of encryption algorithm that, similarly to MACs, produces an authentication tag in combination with message encryption, and this type is called *authenticated encryption* (AE) algorithms. In other words, a single AE algorithm offers the characteristics of both a good cipher and a MAC.

The cipher and MAC can have three possible combinations in the algorithm, as shown in Figure 3.1, which differ in the order in which the encryption is performed and the authentication tag is generated: *encrypt-and-MAC*, *MAC-then-encrypt*, and *encrypt-then-MAC*.



((a)) Encrypt-and-MAC **((b))** MAC-then-encrypt **((c))** Encrypt-then-MAC

Figure 3.1: Chiper and MAC combinations

In the *encrypt-and-MAC* composition (Figure 3.1(a)) the ciphertext and the authentication tag are generated independently of each other. Given a plaintext P, the sender computes the ciphertext C by computing $C = E(K_1, P)$, where E is the encryption algorithm, while the authentication tag T is generated by the operation $T = MAC(K_2, P)$. This two operations can be performed in parallel or ciphertext and authentication tag first.

Once C and T have been calculated, the sender sends them to the recipient and it calculates the plaintext by decrypting the ciphertext $P = D(K_1, C)$. From it it computes $MAC(K_2, P)$ to compare the authentication tag of the decrypted plaintext with the authentication tag that was received. If C or T are corrupted, the verification fails and the message is considered invalid.

A more secure composition to do this is *MAC-then-encrypt* (Figure 3.1(b)), where the authentication tag $T = MAC(K_2, P)$ is first computed, after which the plaintext and the tag are encrypted together $C = E(K_1, P \parallel T)$. When the recipient receives the ciphertext, it decrypts it to find the plaintext and the tag $P \parallel T = D(K_1, C)$, and as was done in the previous composition it verifies the correctness of the data by calculating the authentication tag from the decrypted plaintext $MAC(K_2, P)$.

The greater level of security is given by the fact that with encrypt-and-MAC, even with a secure MAC, it can lose information on the plaintext, making it easier to recover. In MAC-then-encrypt, however, the tag is encrypted along with the plaintext, thus eliminating any possible leakage that may occur. However, the recipient must decrypt C before being able to verify that the received packet is not corrupted, exposing it to potential corrupted plaintext.

The *encrypt-then-MAC* (Figure 3.1(c)) is the most secure of the three compositions. The sender first computes the ciphertext $C = E(K_1, P)$ and from it calculates the tag $T = MAC(K_2, C)$. In order to verify that it has not received corrupted data, the recipient first computes the authentication tag using the received ciphertext, and by comparing it with the received tag, immediately understands the correctness of the data. In this way it is not necessary to decrypt a potentially corrupted ciphertext to perform this verification. Another advantage is that an attacker cannot send a pair of C and T without breaking the MAC, making malicious activities more difficult.

An alternative to these cipher and MAC combinations can be *authenticated ciphers*, which are like normal ciphers except that they return an authentication tag along with the ciphertext. Encryption in this type of cipher is represented as $AE(K, P) = (C, T)$, where AE stands for authenticated encryption, while decryption is represented as $AD(K, C, T) = P$, where AD stands for authenticated decryption. As can be seen, the work that is done is the same as the MAC and cipher combinations, with the advantage that it is simpler, faster, and often more secure.

There are situations where needs to be sent, along with the encrypted message, a header to the packet that contains information about it that must be read, and therefore must not be encrypted, but at the same time it is important to ensure that it has not been modified, and therefore must be authenticated along with the rest of the message. In cryptography this header is called *associated data*, and to achieve this goal the notion of *authenticated encryption with associated data*

(AEAD) was created. In this type of algorithms it is possible to attach a cleartext data together with the ciphertext, so that if the cleartext data is corrupted, the authentication tag will not validate it and therefore the ciphertext will not be decrypted.

The operation that is done in this case can be written as $AEAD(K, P, A) = (C, A, T)$, where the associated data A is kept untouched, and the tag T depends on both the plaintext P and A. Similarly, decryption is performed by $ADAD(K, C, A, T) = (P, A)$. One peculiar thing to note is that both P and A can be left empty, and in the first case AEAD will become a simple MAC, while in the second it will become a normal authenticated cipher.

In order to avoid the predictability of encryption, and thus return different ciphertexts by repeatedly encrypting the same plaintext, the operation is done using an additional parameter, the *Initial Value* (IV), or *nonce* (a number that can be used only once). So the authenticated encryption can be expressed as $AE(K, P, A, N)$, where it is up to the encryption operation to choose the nonce N that has never been used together with that same key. Similarly, decryption must be performed using the same nonce, and can be expressed as $AD(K, C, A, T, N) = (P, A)$ [1].

Following a long tradition of competitions focused on finding the best cipher in a given domain, as was done for the Advanced Encryption Standard, in 2013 a group of international cryptography researchers launched the *Competition for Authenticated Encryption: Security, Applicability, and Robustness* (CAESAR) to encourage the design of authenticated encryption schemes. After 6 years of public scrutiny, the final portfolio was announced in 2019, which is organized into three use cases:

1. Lightweight applications.

2. High-performance applications.

3. Defense in depth.

The first choice for lightweight applications was won by the *ASCON* scheme, designed by Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer [21]. ASCON was subsequently submitted to the NIST lightweight cryptography (LWC) competition, becoming the new NIST lightweight cryptography standard in 2023 [22].

In Section 3.2 the internal structure of the ASCON algorithm will be explained, explaining how encryption and decryption work and which layers form the permutations that are used. The possible cryptanalysis and the attacks that can be launched, explaining the robustness of the algorithm instead are reported in section 3.3.

3.2 Internal structure

ASCON, as presented in the submission to NIST in [23], is a cipher suite that provides authenticated encryption with associated data (AEAD) and hashing functionality. The authenticated ciphers that make up the suite are ASCON-128 and ASCON-128a, which provide 128-bit security and internally use the same 320-bit permutation. This suite was designed to have great security and robustness, but at the same time occupy little area in terms of hardware so that it can be used in applications where there are resource restrictions.

For authenticated encryption, these types of algorithms are parameterized by the key length $k < 160$ *bits*, the rate (data block size) r, and the number of internal rounds a and b. The authenticated encryption procedure is represented by $AE(K, N, A, P) = (C, T)$, where in input we have K, that is the secret key with k bits, N, that is the nonce with 128 bits, A, that is the associated data that can have an arbitrary length, and P, that is the plaintext also of an arbitrary length. As output of this procedure we have C, which is the authenticated ciphertext of the same length as the plaintext P, and an authentication tag T of length 128 bits.

The decryption procedure instead is represented by $AD(K, N, A, C, T) \in \{P, \perp\}$, where K is the key, N is the nonce, A is the associated data, C is the ciphertext, and T is the tag, providing as output the plaintext P if the tag verification is successful, or $\perp$ if it fails.

Table 3.1 shows the recommended parameters in [23] for authenticated encryption instances. The table is ordered by priority: the first recommendation is ASCON-128, and the second is ASCON-128a.

Name	Bit size of				Rounds	
	key	nonce	tag	data block	p^a	p^b
ASCON-128	128	128	128	64	12	6
ASCON-128a	128	128	128	128	12	8

Table 3.1: Parameters for recommended authenticated encryption schemes

The ASCON algorithm uses a 320-bit state S that is updated through the permutations p^a and p^b which are formed by a rounds and b rounds respectively. The state S is divided into an outer part S_r of r bits and an inner part S_c of c bits. The value of the rate r and the capacity $c = 320 - r$ depends on the ASCON variant.

In round transformations the state S is divided into five 64-bit register words x_i, as illustrated in figure 3.2.

$$S = S_r \parallel S_c = x_0 \parallel x_1 \parallel x_2 \parallel x_3 \parallel x_4 \tag{3.1}$$

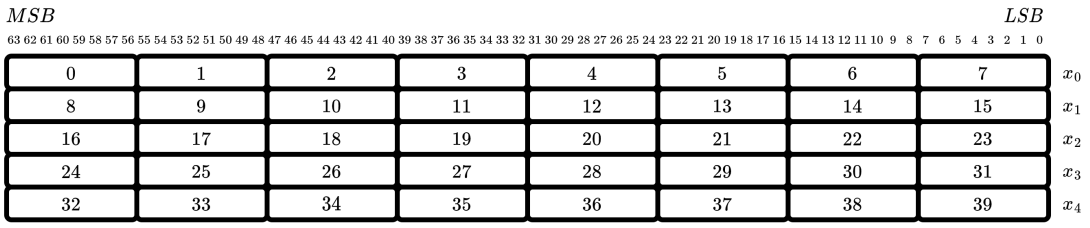
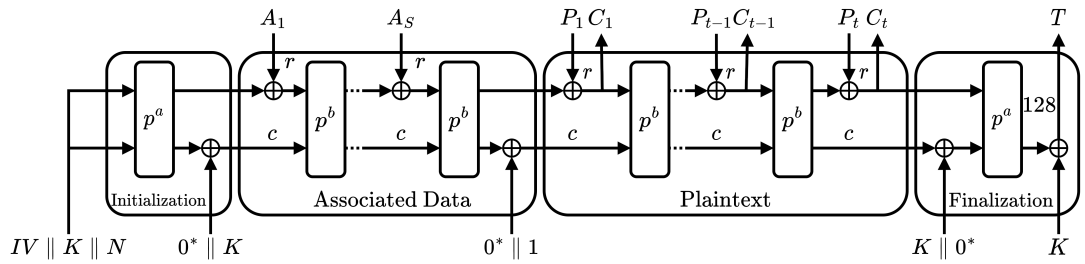


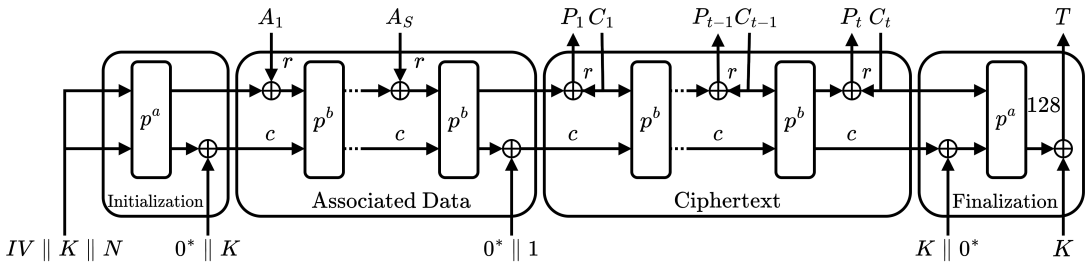
Figure 3.2: Interpretation of the state as a byte-array.

As can be seen from Figure 3.2, when S is interpreted as a byte array it starts with the most significant byte of x_0 as byte 0 and ends with the least significant byte of x_4 as byte 39.

3.2.1 Authenticated Encryption



((a)) Encryption.



((b)) Decryption.

Figure 3.3: ASCON's mode of operation

The mode of operation of ASCON for authenticated encryption is based on *duplex mode*. The duplex construction is often used in *sponge mode*, which is a symmetric-key operation mode based on a fixed permutation as the underlying

primitive. The duplex construction uses a fixed permutation to make plaintext absorption and ciphertext squeeze occur at the same rate as the sponge construction. This allows for a simple and lightweight mode that requires neither key scheduling nor an implementation for the decryption algorithm. The basic design principle of duplex mode is to use the upper layer of the sponge construction of the output of the previous permutation to be combined with the plaintext to generate the ciphertext and the input of the upper layer for the next permutation, while the lower layer of the output of the previous permutation is directly passed as the input of the lower layer of the next permutation [24].

The encryption operation of ASCON, as illustrated in Figure 3.3(a), is divided into 4 steps:

- *Initialization*, where the initial state is formed;
- *Associated Data*, where the associated data is processed;
- *Plaintext*, where the plaintext is processed;
- *Finalization*, where the tag is generated.

The decryption operation has the exact same steps except for the third one, which is modified to obtain the plaintext, as illustrated in Figure 3.3(b).

Initialization

The *initial state* is generated by combining the secret key K, the 128-bit nonce N, and an initial value IV, as shown in equation 3.2.

$$S \leftarrow IV_{k,r,a,b} \parallel K \parallel N \tag{3.2}$$

The IV specifies the length of the key k, the rate r, the number of rounds a for initialization and finalization, and the number of intermediate rounds b, all expressed as 8-bit integers. The IV used in ASCON-128 and ASCON-128a are shown in equation 3.3.

$$IV_{k,r,a,b} \leftarrow k \parallel r \parallel a \parallel b \parallel 0^{160-k} = \begin{cases} 80400c0600000000 & \text{for ASCON-128} \\ 80800c0800000000 & \text{for ASCON-128-a} \end{cases} \tag{3.3}$$

The state S, in the initialization step, is transformed using a rounds of the permutation p, after which it is XORed with the secret key.

$$S \leftarrow p^a(S) \oplus (0^{320-k} \parallel K) \tag{3.4}$$

Processing Associated Data

To process the associated data ASCON divides it into s blocks of r bits $A_1, ..., A_s$. In case the associated data is not a multiple of r, a padding is performed by adding a single 1 and the minimum number of 0s to achieve this condition.

Each block A_i is XORed with the first r bits of state S, followed by b rounds of the permutation p of the resulting state S.

$$S \leftarrow p^b((S_r \oplus A_i) \parallel S_c), \quad 1 \leq i \leq s \tag{3.5}$$

After processing all associated data blocks, the S state is xored with a 1-bit domain separation constant.

$$S \leftarrow S \oplus (0^{319} \parallel 1) \tag{3.6}$$

If there is no associated data to process, this entire step is omitted.

Processing Plaintext/Ciphertext

As in the case of associated data, the plaintext P is also split into t blocks of r bits $P_1, ..., P_t$, using the same padding technique to have the length of the plaintext a multiple of r.

In the case where the encryption operation is being performed, each block of the plaintext P_i is XORed with the first r bits of the state S, followed by the extraction of the ciphertext block C_i. Then b rounds of the p transformation are performed before the next block is extracted. This operation is done for all blocks except the last one, where the permutation is omitted.

$$C_i \leftarrow S_r \oplus P_i \tag{3.7}$$

$$S \leftarrow \begin{cases} p^b(C_i \parallel S_c) & \text{if } 1 \leq i < t \\ C_i \parallel S_c & \text{if } i = t \end{cases} \tag{3.8}$$

In order to have a ciphertext of the same length as the plaintext, the last block of the ciphertext C_t is truncated to the length of the last block of the unpadded plaintext.

In the case of the decryption operation, the plaintext block P_i is calculated by doing an XOR between the first r bits of the state S and the ciphertext block C_i. After that, the first r bits of the state S are replaced by the ciphertext block C_i and, as in the case of encryption, b rounds of the permutation p are performed.

$$P_i \leftarrow S_r \oplus C_i \tag{3.9}$$

$$S \leftarrow p^b(C_i \parallel S_c), \quad 1 \leq i < t \tag{3.10}$$

Finalization

As a final step, for finalization the secret key K is XORed with the internal state and then the result is transformed through a round of permutation p. Finally the tag T is generated by XORing the last 128 bits of the state with the last 128 bits of the key K.

$$S \leftarrow p^a(S \oplus (0^r \parallel K \parallel 0^{c-k})) \tag{3.11}$$
$$T \leftarrow \lceil S \rceil^{128} \oplus \lceil K \rceil^{128} \tag{3.12}$$

The encryption algorithm returns the tag T along with the computed ciphertext. The decryption algorithm returns the plaintext only if the computed tag is equal to the tag that was received.

3.2.2 Permutation

The main transformations that are performed in ASCON schemes are the two *permutations* p^a and p^b. They iteratively execute rounds of *substitution-permutation network* (SPN) based transformations that consist of three steps:

- p_C, *addition of constants*;
- p_S, *substitution layer*;
- p_L, *linear diffusion layer.*

p^a and p^b differs only on the number of rounds executed.

As mentioned at the beginning, the 320-bit state S is divided into five 64-bit register words x_i, $S = x_0 \parallel x_1 \parallel x_2 \parallel x_3 \parallel x_4$.

Addition of Constants

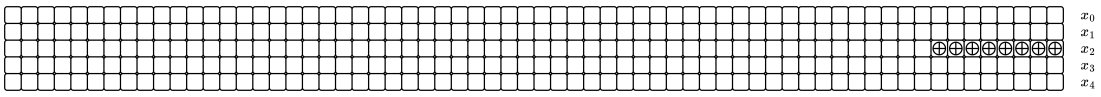


Figure 3.4: Round constant addition p_C.

The *constant addition* layer p_C adds a *round constant* c_r to the register word x_2 of state S of round i, as shown in Figure 3.4.

$$x_2 \leftarrow x_2 \oplus c_r \tag{3.13}$$

Both indices r and i start from 0 and for p^a we use $r = i$ while for p^b is $r = i + a - b$. The round constants used in each round for p^a and p^b are shown in Table 3.2.

p^{12}	p^8	p^6	Constant c_r
0			00000000000000000f0
1			00000000000000000e1
2			00000000000000000d2
3			00000000000000000c3
4	0		00000000000000000b4
5	1		00000000000000000a5
6	2	0	0000000000000000096
7	3	1	0000000000000000087
8	4	2	0000000000000000078
9	5	3	0000000000000000069
10	6	4	000000000000000005a
11	7	5	000000000000000004b

Table 3.2: The round constants c_r used in each round i of p^a and p^b

Substitution Layer

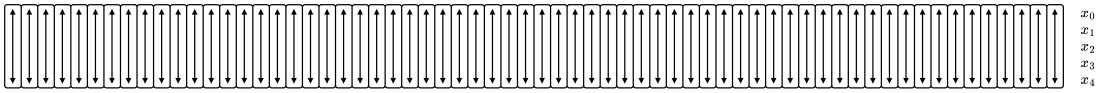


Figure 3.5: Substitution layer p_S with 5-bit S-box.

In the *substitution layer* p_S, the S state is transformed using 64 5-bit S-boxes, showed in Figure 3.6, to replace each bit-slice of the five registers $x_0, ..., x_4$, as shown in Figure 3.5. The lookup table of this S-box is given in Table 3.3, where the x_0 bit is used as the MSB, while the x_4 bit is used as the LSB.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	04	0b	1f	14	1a	15	09	02	1b	05	08	12	1d	03	06	1c
10	1e	13	07	0e	00	0d	11	18	10	0c	01	19	16	0a	0f	17

Table 3.3: ASCON 5-bit S-box lookup table.

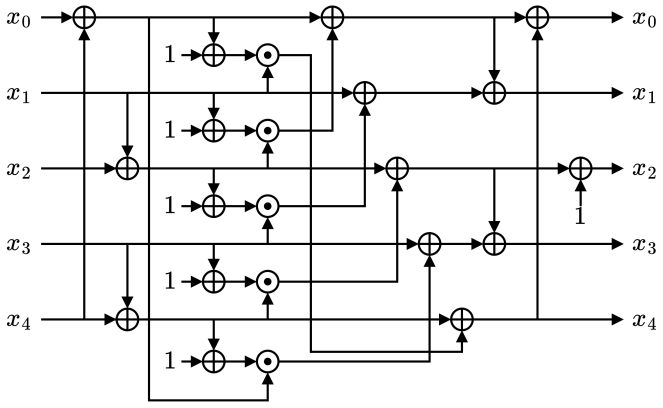


Figure 3.6: ASCON 5-bit S-box.

Linear Diffusion Layer

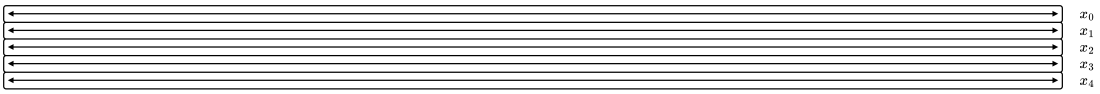


Figure 3.7: Linear layer with 64-bit diffusion functions.

The *linear diffusion* layer p_L, showed in Figure 3.7, applies diffusion using for each 64-bit register word x_i the linear function:

$$\begin{aligned}
x_0 &\leftarrow \textstyle\sum_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \\
x_1 &\leftarrow \textstyle\sum_1(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \\
x_2 &\leftarrow \textstyle\sum_2(x_2) = x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6) \\
x_3 &\leftarrow \textstyle\sum_3(x_3) = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17) \\
x_4 &\leftarrow \textstyle\sum_4(x_4) = x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)
\end{aligned} \tag{3.14}$$

3.3 Cryptanalysis

As seen in Chapter 2, there are several types of cryptanalysis that can be used to break a cipher. Two of the most common forms of cryptanalysis are linear and differential cryptanalysis. These types of attacks exploit the predictable relationship between input and output differences (differential cryptanalysis) or correlations (linear cryptanalysis) for this very purpose. Although the ASCON permutation

design is based on two lightweight operations with individually non-ideal linear and differential properties, when put together they achieve good combined properties against these types of cryptanalysis.

3.3.1 Differential Cryptanalysis

In *differential cryptanalysis*, an attacker searches for patterns that cause specific differences in input to produce certain differences in output. Finding a relationship that has a high probability allows the attacker to eventually find the secret key.

ASCON is designed to minimize this probability. It does this by using the S-box, which allows for confusion in the data. In the algorithm's submission to NIST competition [23], it is reported that the maximum differential probability is 2^{-2}, which means that it is very unlikely that a difference in input will generate a predictable difference in output.

Also the differential *branch number* is 3. It is a numeric value that characterizes the amount of diffusion introduced by a boolean function that maps an input vector to an output vector. A high branch number value means better resistance against differential cryptanalysis, this because a small change in the input would produce a large change in the output and to obtain a small change in the output a large change in the input is needed [25]. While this value is smaller than the one obtained in the case of AES, where the branch number is equal to 5, it is large enough to resist these types of cryptanalysis.

In SPN structures there is a strong relationship between the differential probability and the number of differential active S-boxes. When the number of differential active S-boxes is high the value of the differential probability is low, on the contrary when the number of active S-boxes is low the value of the differential probability is low. An S-box is considered differentially active when it receives a non-zero difference as input. When an S-box is active, it introduces confusion between input and output, so the aim of every cipher designer is therefore to maximize the number of active S-boxes. In the case of ASCON the number of active S-boxes of 3 rounds is 15, while the number increases to 78 if it is calculated for 5 rounds. This exponential growth in active S-boxes across rounds makes cryptanalysis highly impractical as it becomes increasingly difficult to find exploitable differential features.

3.3.2 Linear Cryptanalysis

The second type of attack is *linear cryptanalysis*, which exploits the linear correlation between the input and output bits. Again, the strength of ASCON against this type of attack can be traced back to the S-box, which destroys any possible correlation that can occur.

Also in the document for the submission to NIST competition, it can be read

that this algorithm has a maximum linear bias of 2^{-2}, which indicates how strong a correlation between input and output can be, and being very low means that finding a linear correlation between input and output has a very low probability. Also in this case the linear branch number is equal to 3, so each bit in output depends on at least 3 bits in input, thus making it very difficult for an attacker to trace a linear pattern to exploit.

As with differential cryptanalysis, the number of active S-boxes increases with the number of rounds, from 13 for 3 rounds to 67 for 5 rounds. This makes it extremely difficult for an attacker to discover any linear relationship across multiple rounds of the cipher.

3.3.3 Side-Channel Cryptanalysis

When designing ASCON, the algorithm was made with a primary goal of having a natural protection against *side-channel attacks*. For this purpose it is necessary that the S-box is easy to protect, so the one designed has a low algebraic degree of 2 and a low number of boolean multiplications that make it possible to use Threshold Implementations or similar approaches. The algebraic degree of a function measures the complexity of its polynomial representation. Since that of ASCON is equal to 2 it makes it easier to decompose into simple polynomials. The Threshold Implementation technique is a countermeasure against side-channel attacks in which the operations of a given algorithm are divided into multiple shares, such that none of them reveal any sensitive information. Since functions with low algebraic degree are easier to decompose, this helps the implementation of this technique. Additionally, since it has few Boolean multiplications, this makes the Threshold Implementation efficient given its low computational cost and power consumption.

But this is not the only defense against such attacks. Compared to other sponge-based authenticated algorithm designs, ASCON uses stronger key initialization and key finalization phases, which means that even if an attacker manages to extract the internal state during data processing, it is not directly linked to the secret key, preventing its recovery [23].

Chapter 4

Power Analysis Side Channel Attack

4.1 Introduction

During the execution of an algorithm we can collect some information about what it is doing by looking at its power consumption. The power analysis attack is based on this principle.

The building block of a digital system is the transistor, and in present days the most used gate technology is the FinFET. FinFET is the improved version of CMOS transistors and presents a less power dissipation and propagation delay. The *power consumption* in this type of transistors can be expressed with the expression in 4.1 [26].

$$P_{total} = P_{short} + P_{static} + P_{dynamic} \tag{4.1}$$

The first dissipation source is due to the so-called *"short-circuit current"*. This current is due to the fact that exists a short period during the switching of a gate in which the nMOS and the pMOS are conducting simultaneously. The second one is due to the *leakage currents* in transistors. FinFET transistors have an improved gate control with respect CMOS, and so this current is significantly reduced. The last one instead is due to *charge and discharge of the load capacitance*. This source of dissipation is particularly relevant from a side-channel point of view because it correlates the device's internal data and the power consumption. It can be written as the equation in 4.2, where C_L is the load capacitance, V_{DD} is the voltage of the power supply, f is the frequency at which the device is running, and E_{SW} gives the information of the percentage of times the signal changes..

$$P_{dynamic} = C_L V_{DD}^2 f E_{SW} \tag{4.2}$$

The power consumption depends on what the device is doing: if the device is in an idle configuration, it will have a lower consumption compared to the one it has if it is doing a multiplication, and a multiplication consumes more than a sum. Different executions of the same operation can also be distinguished by looking at the power trace alone, making these devices weak from a security point of view (a trace refers to a set of power consumption measurements taken across an operation).

To measure this power consumption a simple configuration can be build. A *small resistor*, typically in the order of 50 Ω, is inserted in series to the power supply or ground input. By measuring the voltage across the resistor, the current absorbed or released by the circuit can be determined, which allows observation of the power consumption at that instant. The equipment to make this type of measurement is not very expensive. With less than $400, devices capable of sampling at 100 Msps and transferring the data to the PC can be purchased.

There are different types of power analysis attack, each more effective than the previous one but more expensive to execute them. In Section 4.2 is explained the Simple Power Analysis, in Section 4.3 Differential Power Analysis, and in Section 4.4 Correlation Power Analysis. Finally in Section 4.5, some countermeasures that can be adopted to improve security against these types of power analysis are investigated in more detail.

4.2 Simple Power Analysis (SPA)

Simple Power Analysis (SPA) [26] is a technique that involves directly interpreting power consumption measurements collected during cryptographic operations.

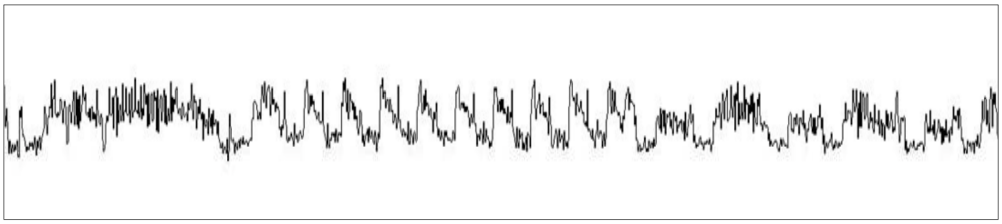


Figure 4.1: SPA monitoring from a single AES encryption performed by a smart card

In figure 4.1 we can see a power trace taken during an AES encryption operation on a smart card. The AES-128 algorithm to encrypt (or decrypt) the plaintext is realized by running 10 rounds. Visually we can notice in the middle of the trace that we have ten times the repetition of the same power figure, so probably the execution of the algorithm is performed in that period of time. This information is not very useful to find the key, but it may be a preliminary step in a more powerful

attack.

There are cases instead that this sequence of operations can provide useful information, mainly when the instruction flow depends on the data. An example is the code in Algorithm 2.

Algorithm 2 Simple password check function pseudocode

```
1: function passwordCheck(userPassword)
2:     secretPassword ← 1, 2, 3, 4
3:     errorLedOff()
4:     if length(secretPassword) != length(userPassword) then
5:         errorLedOn()
6:         return 0
7:     end if
8:     for position p of the password in range {0, ..., length(secretPassword) - 1}
   do
9:         if secretPassword[p] != userPassword[p] then
10:            errorLedOn()
11:            return 0
12:        end if
13:    end for
14:    return 1
15: end function
```

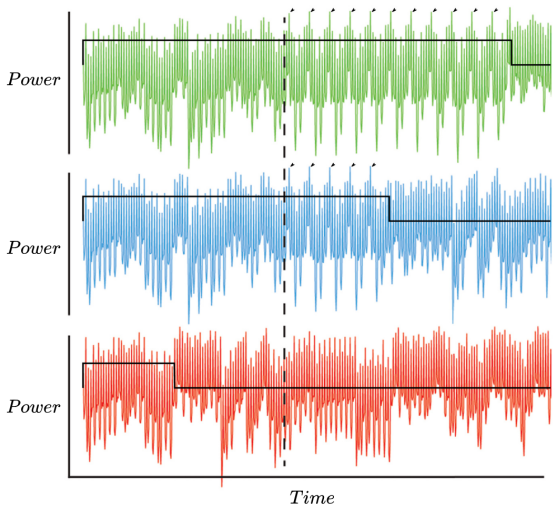


Figure 4.2: Example of power traces for correct, partially correct, and incorrect passwords.

In figure 4.2 [27] we have some power traces collected executing a code similar to that one. The first power trace is the one that we could see if the submitted password is correct. In this case the secret password is eleven digits long, and we can distinguish eleven same patterns after the dashed line, which is the trigger signal that starts the comparison of each character. In the second power trace instead we can see the case for a partially correct password. This time we have only the first five digits that are correct, in fact we see only five repetitions. The *passwordCheck* code checks the length of the submitted password before doing the comparisons, and in fact the last trace shows what happens if a password of only four digits is sent. As we can see, no comparison is done this time, and so no pattern appears.

SPA relies also on the fact that each instruction executed by a microcontroller has a unique power consumption trace. For instance, it is possible to differentiate a multiplication from a load instruction. This type of attack is not very valuable in the case of symmetric encryption, since the power profile is the same regardless of the key. In the case of asymmetric encryption, the various branches of the algorithm very often depend on a single bit of the key, and therefore the power profile varies if the key bit is 0 or 1. For this reason, SPA is often used to attack asymmetric encryption. An example is the RSA algorithm. RSA consist of computing $R = y^x mod n$, where n is public and y can be found. The attacker's goal is to find x. The simple modular exponentiation algorithm is depicted in Algorithm 3 [6][8] which computes $R = y^x mod n$, where x is w bits long.

Algorithm 3 Pseudocode that shows the simple modular exponentiation algorithm

```
1: procedure MODULAREXPONENTIATION(x,y,n)
2:     w ← length of x
3:     s = new array of 1 of length w + 1
4:     R = new array of 1 of length w
5:     for each k in range {0, ..., w} do
6:         if the k bit of x is 1 then
7:             R[k] ← (s[k] * y) mod n
8:         else
9:             R[k] ← s[k]
10:        end if
11:        s[k+1] ← R[k]^2 mod n
12:    end for
13:    return R[w-1]
14: end procedure
```

If the bit of the exponent is 1 is computed the operation $(sk * y) mod n$, otherwise it is not. So if we could have the power trace when the algorithm is executed, and

we know the position in the trace of this operations, we can see if the multiplication is done or not, and from that we can understand whether the key bit is a 0 or a 1.

An example of how can be implemented an attack of this type is showed in Algorithm 4.

Algorithm 4 Simple Power Analysis attack implementation

```
Require: inputList
Ensure: secretKey
▷ Step 1: Preparing for the attack
deviceVictim ← initializeDevice()
oscilloscope ← initializeOscilloscope()
secretKey ← NULL
powerTraces = new array
▷ Step 2: Power trace acquisition
for each input text i in inputList do
    output ← deviceVictim.runOperation(i)
    trace ← oscilloscope.measure()
    append trace to powerTraces
end for
▷ Step 3: Power trace analysis
for each trace t in powerTrace do
    repetitivePattern ← traceAnalysis(t)
    for each pattern p in repetitivePattern do
        if the pattern is equal to a critical operation then
            keyBit ← determineKeyBit(p)
            add key_bit to secretKey
        end if
    end for
end for
```

The first step is dedicated to the *preparation for the attack*. All devices that are to be used need to be initialized, then the oscilloscope is connected and set it up with a suitable configuration and prepare the victim device by loading the algorithm that needs to be broken onto it (which can be implemented either in software or hardware). For example we can imagine to upload on the device the simple modular exponentiation algorithm. All variables that are to be used during the attack must also be initialized.

In the second step the *power traces needed for running the attack are acquired*. For doing so the device victim execute the operation with a set of inputs, in this case the simple modular exponentiation algorithm, and for each of them, with the oscilloscope, the traces are acquired and saved. It can be said that this step is also

a sort of preparation for the attack, which will be launched in the next step.

Once all the data has been acquired, the *actual attack begins*. First, the traces are analyzed to find repetitive patterns that should correspond to cryptographic operations. When the analysis is complete, the repetitive patterns are compared to the patterns of known operations of the cryptographic algorithm, and if a match is found, the bit associated with that particular pattern is derived from it. For example, in the case of the simple modular exponentiation the execution of the two branches has two different power patterns because the key bit in the case of a 1 is different with respect the case of a 0 because an additional operation is performed, so by comparing that pattern with the one it would have if it were a 1, itthe value of the bit can be immediately determin In this way the secret key is discovered bit by bit until the whole key is found.

In general SPA attacks are very simple to prevent [7]. To mask many characteristics of SPA attacks, simply avoid procedures that use secret intermediates or not use keys for conditional branching operations. In addition, most hardware implementations of hard-wired cryptographic algorithms have variations in power consumption that are so small that it is not possible to derive the key.

4.3 Differential Power Analysis (DPA)

Because transistors are small, and a single bit affects only a handful of them, the resulting variation is often small and, because the measurement is not ideal, it is hidden below the noise level. To overcome this problem, statistical techniques can be used: by collecting many power traces and averaging them, it is possible to reduce the noise so as to reveal what is underneath. *Differential Power Analysis* (DPA) is based on this principle.

In a microcontroller we have a bus, and at every clock cycle the data on this line changes. This operation, due to the charging and discharging of the capacitors on the line, also brings with it current peaks, from which it is possible to determine the number of ones in the data being processed. This number is referred to as the *Hamming weight* [27]. For example the Hamming weight of the hexadecimal number 0xA4 is 3 because in binary it can be written as 10010100 and in it there are three ones. Hamming weight and power spikes are strictly dependent, and so we can use it to obtain the data that is being processed on the bus.

Let's consider a simple circuit in which the input is XORed with an 8-bit secret key, and after that the output is passed through a known lookup table that map every byte with another value without having access to its encrypted output. We want to break the least significant bit (LSB) of the key. To do this, we predict the internal state for each possible key value and split the power traces we collected based on what we get. If the split is done correctly with the correct key guess,

since ones consume a different amount of power than zeros, there's a measurable difference. With DPA attacks we can only get a hint about the possible value of the key, without finding out how the key decrypts the data.

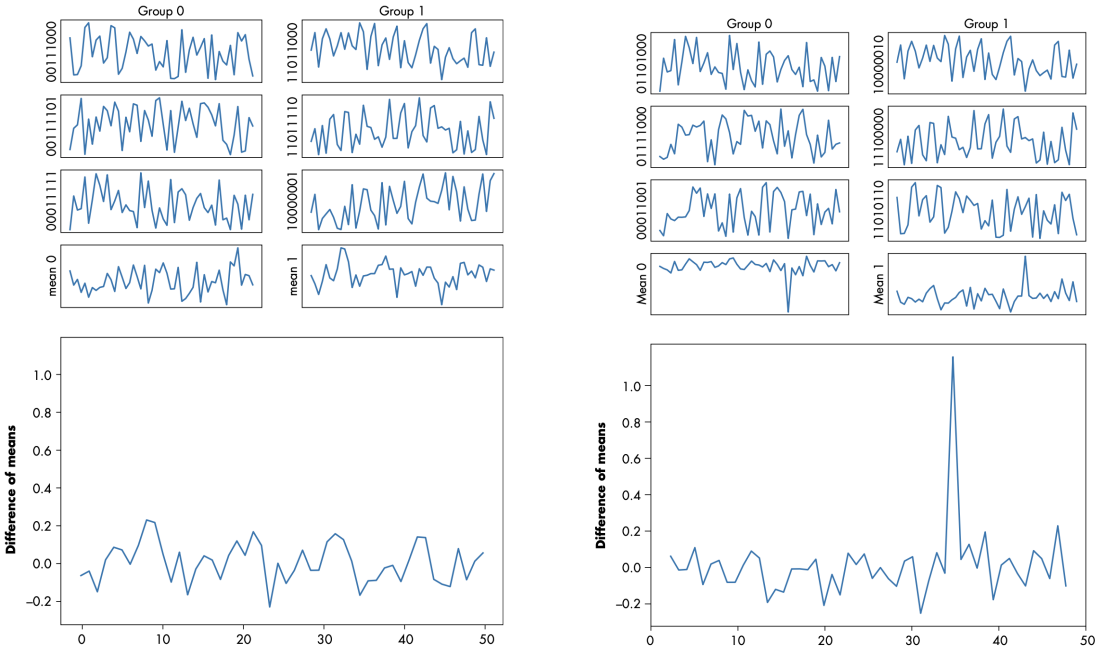
So we get a set of power traces called t and for each of them we associate the input that we give to the system to obtain them using and array called p. Algorithm 5 shows a pseudocode that explains the DPA attack.

Algorithm 5 Pseudocode that computes the output and guesses a single key byte

```
1: diffarray = new array
2: for each key guess i of the secret key in range {0x00, ..., 0xFF} do
3:     zerosarray = new array
4:     onesarray = new array
5:     for each trace d in range {0, 1, ..., D-1} do
6:         calculate hypothetical output h = lookupTable[i ⊕ p[d]]
7:         if the LSB of h = 0 then
8:             append t[d] to zerosarray[]
9:         else
10:            append t[d] to onesarray[]
11:        end if
12:    end for
13:    difference ← mean(onesarray) - mean(zerosarray)
14:    append difference to diffarray[]
15: end for
```

This code lists all possible key bytes to guess, and for each of them loop over all input texts of the recorded power traces. An hypothetical output h is generated using the guessed secret key byte i and the input data p, and for each of them look at the LSB of the hypothetical output and add the corresponding recorded power trace to an array if it is 0, to another array if it is 1. If the guess is incorrect the power traces are randomly splitted in this two groups, and so the mean power consumption of the two is the same. If subtracted the means from each other the result will be noise, as it is depicted in Figure 4.3(a). If instead the guess is correct what is computed hypothetically is the same as the one computed on the device, and so the two groups for the zeros and ones are correctly splitted. If ones and zeros consume a different amount of power, the difference appears if the difference of means is done, as it is depicted in Figure 4.3(b).

The *difference of means* stands for taking the difference of the two groups averaged, and this operation is done at line 13 in Algorithm 5. This method works because the noise is uniformly distributed between the two groups, so when a sufficient number of traces is averaged any other contribution cancels out and only the one from the byte chosen for the division remains.



((a)) Incorrect guess

((b)) Correct guess

Figure 4.3: Averaging many traces into ones and zeros

As an example of an implementation for a differential power analysis attacks we can see at the pseudocode in Algorithm 6. In Figure 4.4 can be seen a graphical representation of the attack for a better understanding of the steps that need to be performed.

The first two steps are identical to those we had for a SPA attack: first there is the preparation for the attack, and so the device and the oscilloscope are initialized, and after that the power traces are acquired when the device runs the algorithm to be attacked.

In the last step there is the power trace analysis to find the values of each subkey of the key. As we can see in the code, we have a first for loop that loops through all the bytes of the key. For example, if we have a key that is 16 bytes long, in this for loop all the bytes are analyzed individually.

After that we have a second for loop that this time instead loops for all the values that the subkey can assume, and so all 256 values that it can assume are analyzed. As we saw before, we now need a way to split the tracks into two groups, depending on the bit obtained in the internal state with that particular subkey value, and from these two groups calculate the difference of the means. So first it is calculated the hypothetical output using the subkey and the input that was provided for the power trace, and then, depending on whether the LSB in question is a 0 or a 1, the power trace for that input is put into one or the other group. This

Algorithm 6 Differential Power Analysis attack implementation

Require: inputList
Ensure: keyGuess

```
▷ Step 1: Preparing for the attack
deviceVictim ← initializeDevice()
oscilloscope ← initializeOscilloscope()
powerTraces = new array
secretKey = new array
▷ Step 2: Power trace acquisition
for each input text i in inputList do
    output ← deviceVictim.runOperation(i)
    trace ← oscilloscope.measure()
    append trace to powerTraces
end for
▷ Step 3: Power trace analysis
for each byte b of the secret key in range {0, 1, ..., B-1} do
    maxDiffs = new array of length 256
    fullDiffs = new array of length 256
    for each guess g of the subkey in range {0x00, ..., 0xFF} do
        onesarray = new array
        zerosarray = new array
        for each trace t in range {0, 1, ..., D-1} do
            calculate hypotetical leakage h=lookupTable[g ⊕ inputList[t][b]]
            if the LSB of h = 0 then
                append powerTraces[t] to zerosarray[]
            else
                append powerTraces[t] to onesarray[]
            end if
        end for
        fullDiffTrace ← mean(onesarray) - mean(zerosarray)
        maxDiffs[g] ← max(fullDiffTrace)
    end for
    sortedArgs ← argsort(maxDiffs)
    append sortedArgs[0] to secretKey
end for
```

is done for all the power traces, and once the two groups are populated, they are averaged and then the average of the 1s group is subtracted from the average of the 0s groups.

Once the difference of the means has been obtained, the next operation is to find the maximum value obtained, this is because the parameter *fullDiffTrace* is a vector where each value is the difference of the means for that point in the power trace.

This operations are done for each guess of the subkey, obtaining at the end a vector of the maximum values obtained. Then we proceed to order the arguments of these values from the greatest to the least, as each index is the actual subkey guess. The first value of this sorted values hopefully is the secret byte of the key.

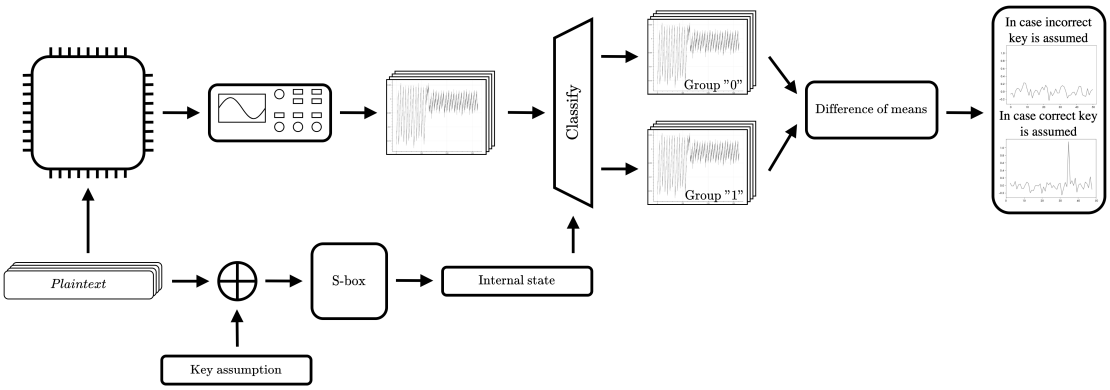


Figure 4.4: Graphical representation of a Differential Power Analysis attack

Experience teaches that attacks of this type run into a very serious problem since some peaks appear even for wrong guesses, those which are called *"ghost-peaks"* [28]. This problem can appear in cryptographic algorithms that use S-boxes for example, since the distribution of the S-box outputs for two different hypotheses are deterministic and thus can be partially correlated. This partial correlation can create false spikes in the difference of means, making it more difficult to distinguish the correct key guess from the incorrect ones. For example, if we have two key hypotheses, $0b0110$ which is the correct key and $0b0101$, if the S-box output for these two key hypotheses is partially correlated due to its structure, doing the difference of means would create a peak not only for the first key hypothesis, which is the correct one, but also for the second one, even if it is incorrect. This peak is called a ghost-peak because it should not occur.

The problem of "ghost-peaks" becomes even more relevant when the peak of the correct hypothesis becomes smaller than the false peaks. DPA implicitly considers the bit of a word independent from each other, and therefore the analysis does not take into account possible correlations that may exist between them. In the real

world, however, the values of the bits of the same word are often deterministically bonded, and this interaction can influence the power consumption during encryption, ensuring that their contribution is not canceled out. This phenomenon can lead to a distortion of the power profile, resulting in the correct peak being narrowed, making it harder to locate, while irrelevant peaks may be enhanced, leading to false positives.

A solution to get a more efficient attack and to get rid of the "ghost-peaks" can be to increase the number of traces acquired. The more data is added, the more the problem is reduced, without however completely removing the "ghost-peaks".

A more advanced method could be to *window* the input data. Looking at the peaks collected, it can be noticed that the correct peak always comes after a certain number of samples, so the loop in the Algorithm 6 can be modified to look for the peak after this point.

Noise is also a problem, and there are several sources that could introduce it into DPA measurements. Some of them could be electromagnetic radiation, thermal noise and quantization errors. The more traces are averaged, the more what is not the DPA signal will be canceled out. Figure 4.5 shows this effect: on the left is depicted the difference of means on 1,000 traces, on the right the one on 100,000 traces. As we can see the random noise is further reduced with a greater number of measurements, and also the peak becomes much more pronounced.

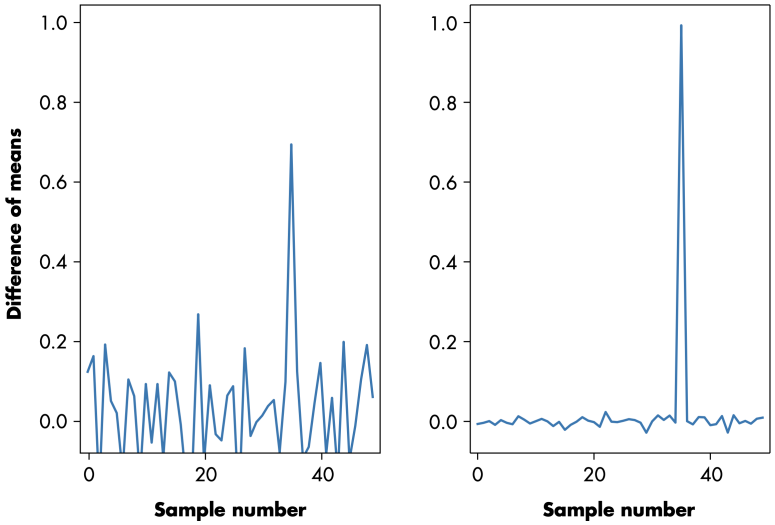


Figure 4.5: Difference of means on 1,000 (left) vs. 100,000 (right) traces

There are generally three types of techniques for preventing DPA attacks [7]:

- The first one consist on *reducing the signal sizes*. This technique can be performed for example using constant execution path code, choosing operations

that leak less information in their power consumption and balancing Hamming Weights. Unfortunately, all this leads to an increase in the size and cost of the device, without completely canceling the size of the signal, and therefore only increases the number of samples the attacker needs to perform the attack.

- The second technique consists of *introducing noise* into the measurement of power consumption. Like the previous technique, also in this case it increases the number of samples useful for carrying out the attack, without eliminating the possibility.

- The last one is to *prevent the attacker from collecting a large number of traces*. This can be done using exponent and modulus modification processes in public key schemes. Not having many traces to average to reduce noise makes the DPA attack ineffective.

4.4 Correlation Power Analysis (CPA)

Correlation Power Analysis (CPA) is a more advanced attack with respect to DPA. It was introduced by Eric Brier, Christophe Clavier, and Francis Olivier in the CHES 2004 paper "Correlation Power Analysis with a Leakage Model" [28]. With DPA the basic idea is to make an assumption about the value of the key, and with that assumption a prediction is made about the value that a bit should have. With CPA on the other hand, the same assumption about the key is always done, but the prediction is based on the value of an entire word. DPA is based on the Hamming weight model, which is the number of bits that are asserted in a word D. If D contains m independent and uniformly distributed bits, this word has an average Hamming weight $\mu_H = \frac{m}{2}$ and a variance $\sigma_H^2 = \frac{m}{4}$.

CPA instead is based on the *correlation factor*. In the paper by Eric Brier, Christophe Clavier and Francis Olivier [28] they proposed an approach based on the *Hamming distance* model, that is a generalization of the Hamming weight model. They used the CPA to identify the parameters of the leakage model.

The reference state from which the bits start is assumed to be a constant machine word R, unknown but not necessarily zero. In general it is assumed that the power leakage due to the data depends on the number of switching bits from one state to the other, which can be described by the Hamming distance between D and R $H(D \oplus R)$. If D is a uniform random value, so is $D \oplus R$, and so $H(D \oplus R)$ has the same mean $\frac{m}{2}$ and variance $\frac{m}{4}$ as $H(D)$.

If we want to calculate the power consumption W for the data dependency, the basic model to do so is shown in equation 4.3, where a is a scalar gain between the Hamming distance and the power consumed, and b encloses all the other variable in the power consumption- e.g. offsets, time dependent components and noise.

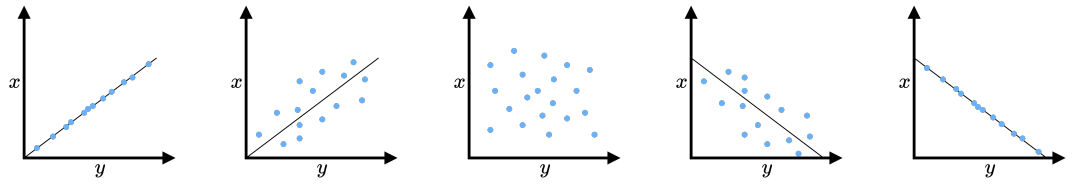
$$W = aH(D \oplus R) + b \tag{4.3}$$

It is also assumed that there is a linear relationship between the power consumption and $H(D \oplus R)$, which implies that the relationship between the variances of the terms used to calculate the power consumption is $\sigma_W^2 = a^2\sigma_H^2 + \sigma_b^2$.

CPA exploits the *correlation factor* ρ_{WH}. It measures the linear relationship between samples of two random variables, in this case the Hamming distance and the measured power traces, and can be computed with the equation in 4.4.

$$\rho_{WH} = \frac{cov(W,H)}{\sigma_W \sigma_H} = \frac{a\sigma_H}{\sigma_W} = \frac{a\sigma_H}{\sqrt{a^2\sigma_H^2 + \sigma_b^2}} = \frac{a\sqrt{m}}{\sqrt{ma^2 + 4\sigma_b^2}} \tag{4.4}$$

The value of this factor is between $-1 \leq \rho_{WH} \leq 1$. It is 1 if the Hamming distance and the power traces are perfectly linear related (Figure 4.6(a)), so the greater the power consumption the higher the hamming distance. It is -1 if they are perfectly negatively correlated (Figure 4.6(e)), so the greater the Hamming distance the lower the power consumption. There is no linear relation if the correlation factor is 0 (Figure 4.6(c)), so it means that for a certain guess the measured trace doesn't correspond at all to the Hamming distance.



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

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

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

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

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

Figure 4.6: Graphical representation of the correlation factor for different relationships between the two variables.

This relationship shows that the correlation factor is maximized when the noise variance σ_b is minimum. Consequently, ρ_{WH} can be used to determine the reference state R. Given a set of data D and their related power consumption W, all the 2^m possible combinations of R can be scanned, and by making a ranking based on the correlation factor they have when combined with the observation W one can make a guess as to which R is more likely. Indeed, if we have a candidate value R' that differ from the true reference R by k bits, the correlation factor will be the one in the equation 4.5.

$$\rho_{WH'} = \rho_{WH}\frac{m - 2k}{m} \tag{4.5}$$

If only one bit is wrong against an 8-bit word, the correlation is reduced by $\frac{1}{4}$. If all the bits are wrong, and so $R' = -R$, in that case the correlation will be $\rho_{WH'} = -\rho_{WH}$. This property is called anti-correlation, and it generates the strongest negative correlation value, equal in absolute value to that of R. So there is no value of R' that can have a correlation value greater than R.

This computation is not so expensive in the case of an 8-bit micro-controller since only 256 values need to be tested. If the number of bits increases this operation may become unsustainable, but in those cases one can work with partial correlation. For further information, see [28].

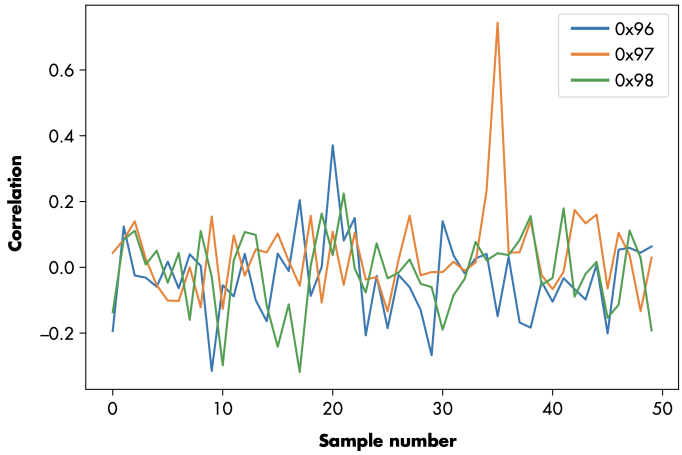


Figure 4.7: Correlation plot of correct key guess and two incorrect key guesses in a CPA attack on AES-128

In figure 4.7 is showed an example of a plot of the correlation for three different key guess in a CPA attack on AES-128 using as leakage model the Hamming weight [27]. As can be seen, between the three plot, the orange one have the larger peak, so the value of that key is expected to be the correct one.

The implementation of a CPA attack is very similar to the ones we have seen previously for this category of Power Analysis Attacks. In Algorithm 7 can be seen the differences in the pseudocode for this case if wanted to attack the AES algorithm. In Figure 4.8 shows a graphical representation of the attack for a better understanding of the steps that need to be performed.

The first two steps are identical. They are the preparation for the attack with the power trace acquisition. What changes in this type of attacks is the third step, the power trace analysis. This time the analysis is based on the leakage model and on the correlation between it and the actual power consumption.

Looking in more detail at the operations that need to be done, first of all the mean and standard deviation of the power traces that have been acquired are calculated. This data will be used later to obtain the correlation. Next, the

Algorithm 7 Correlation Power Analysis attack implementation

Require: inputList
Ensure: keyGuess

```
▷ Step 1: Preparing for the attack
deviceVictim ← initializeDevice()
oscilloscope ← initializeOscilloscope()
powerTraces = new array
secretKey = new array
▷ Step 2: Power trace acquisition
for each input text i in inputList do
    output ← deviceVictim.runOperation(i)
    trace ← oscilloscope.measure()
    append trace to powerTraces
end for
▷ Step 3: Power trace analysis
bestGuess = new array of key length
meanPowerTraces ← mean(powerTraces)
stdDevPowerTraces ← stdDev(powerTraces, meanPowerTraces)
for each byte b of the key in range {0, ..., key length} do
    maxCpa ← new array of 0 of length 256
    for each key guess g in range {0x00, ..., 0xFF} do
        hws = new array
        for each input text inText in inputList do
            hammingWeight ← HW(lookupTable(inText[b] ⊕ g))
            append hammingWeight to hws
        end for
        meanHws ← mean(hws)
        stdDevHws ← stdDev(hws, meanHws)
        correlation ← cov(powerTraces, meanPowerTraces,hws, meanHws)/(stdDevPowerTraces × stdDevHws)
        maxCpa[g] ← max(abs(correlation))
    end for
    bestGuess[b] ← argmax(maxCpa)
end for
```

hypothetical power consumption is calculated using the leakage model. This is nothing more than the Hamming weight of the operation performed by the AES algorithm on the S-box. This operation is performed on all the texts given as input for the power consumption acquisition.

Once all the Hamming weights are calculated, the next step is to compute the mean and standard deviation of them which are used, together with the mean and standard deviation calculated previously for the power traces, to calculate the correlation factor using the formula 4.4. The power traces are vectors showing the power consumption at each sample in time, hence the correlation factor is calculated for each sample of the trace. Lastly, the maximum correlation value between all samples is selected.

These operations are performed for each subkey candidate. In the end, the most probable key candidate is the one with the largest correlation factor. These steps are repeated for all the bytes of the key to recover the complete key.

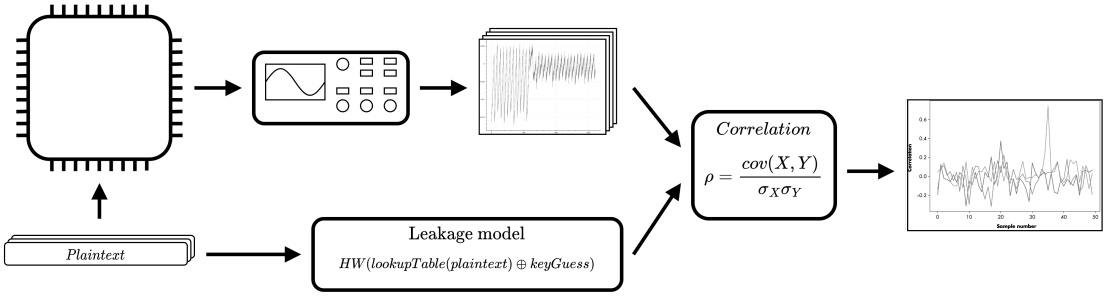


Figure 4.8: Graphical representation of a Correlation Power Analysis attack

The countermeasures that can be exploited to prevent this type of attack are very similar to the ones for the DPA [28]. One countermeasure is to *add additional noise* to the power trace. As for DPA attacks, this operation only increases the number of sample that has to be collected, and can be bypassed averaging. Other countermeasures consist in *desynchronizing* the power traces in the execution of the process, adding for example fake cycles and random delays. Its effects can be corrected applying appropriate signal processing. *Eliminating the correlation between variables and the power trace* is another method, and this can be done by dynamically encrypting the data during the process, for example by encrypting the bus or masking the data.

None of these countermeasures can be considered secure against a potential SCA attack when used alone, but a combination of at least two of the above can be very effective in blocking a potential attacker, thus making the device secure.

4.5 Countermeasures

During the various sections of this chapter, in addition to explaining how the various attacks work, some *countermeasures* have also been proposed to defend against them. There is no definitive countermeasure that can completely eliminate the threat of an attack, but it is possible to make the recovery of sensitive data so difficult as to discourage a potential attacker, minimizing the chances of success. To eliminate the risk of SPA attacks, the solution of *decorrelating* the time between operations and the secret value, and therefore making the execution time constant, is sufficient, but for more advanced attacks, as DPA or CPA attacks, things become considerably more complicated. In order to make an algorithm more secure against this type of attacks, the basic idea is to make the correlation between sensitive data and consumed power as small as possible. However, this is easier said than done.

The simplest thing that can be done is to *add some noise* to the power consumed by running other hardware in parallel. This strategy does not have the objective of decorrelating the signal, but is useful for increasing the cost of the attack by increasing the number of traces needed. To generate this noise in hardware what one can do is run a random number generator or a video encoder on some dummy data while the algorithm is performing the encryption. As previously mentioned, this strategy can be easily circumvented by averaging the results obtained, thus eliminating the noise and obtaining a clean output. Furthermore, in addition to not having a significant improvement in resistance against side-channel attacks, this strategy increases the area occupied by the algorithm if there is no hardware already present that can be exploited to generate the noise.

Another technique that is much more performing but has a big negative impact on the chip area is to use a *dual-rail logic*. Thanks to this type of transmission, each gate and line also has its own negated version, and in this way for each transition from zero to one there is also a transition from one to zero and vice versa, thus balancing the power consumption. This balancing however requires extreme precision so that these transitions occur at the same time, since any imbalances still cause leakage, although much less than in the case without this technique. This strategy must also take into account electromagnetic signals, since depending on how the lines are positioned in space, two inverted signals can amplify or cancel each other out, and therefore increase the chances of success of electromagnetic analysis attacks.

Secret sharing, or *masking*, is another countermeasure that can be adopted. It consists in dividing the sensitive data, i.e. the key and the plaintext in an encryption, into multiple shares, so that each of them does not reveal useful information about the original information.

Example. For example, to divide a piece of data into two shares, two values x_1 and x_2 are chosen so that $x = x_1 \oplus x_2$ results in the original data.

After that, the operations to be performed by the algorithm are adapted so that each cryptographic operation works only on a single share. These operations must be designed to keep the data masked for the entire duration of the execution. Furthermore, to increase the security of this technique, a *mask refreshing* can also be implemented, that is, a periodic regeneration of the shares, this is because some attack techniques can accumulate information over time. Once the encryption is finished, the algorithm outputs an encoded form of the ciphertext, which can be decoded directly thus obtaining the affective ciphertext. This technique is a very effective technique against side-channel attacks, although it is very complex to implement since increasing the number of shares increases the required memory and the computational load, and this can be a limitation in embedded systems where resources are limited.

An alternative to the masking technique is the *hiding* countermeasure. It is based on the fact that operations that are independent of each other can be exchanged randomly so that an attacker cannot know the exact moment in time when it is executed. An example of this technique can be made on the AES S-box, but the same kind of reasoning can be done on the ASCON one. In this algorithm the 16 bytes of the intermediate state are replaced by means of this layer. Instead of doing the byte replacement in parallel, what can be done is to calculate them one at a time in a random order. This reduces the temporal predictability and makes it difficult for an attacker to associate a power consumption with a given instant in time and cryptographic operation.

To conclude this overview of countermeasures that can be implemented, we have seen that these techniques, although they increase the resistance against SCA attacks both individually, but even more if used together, they involve a large cost in terms of area, but especially encryption (or decryption) performance. This is because they add new elements within the cryptographic operation, or modify those already present in order to perform the flow that would be followed in normal calculation in a different way. This thesis therefore focused on a different type of countermeasure, which does not affect the performance of the algorithm, but with a simple change of a specific layer it is possible to obtain satisfactory results regarding the side-channel resistance. This countermeasure uses a different paradigm to achieve the goal, a mathematical paradigm that improves the cryptographic properties in order to achieve the desired results. The layer that has been modified is the *S-box*, and everything will be explained in more detail in Chapter 5.

Chapter 5

S-box countermeasure

5.1 Introduction

One of the main goals of this thesis is to test some lightweight countermeasures on AES and ASCON cryptographic algorithms in order to increase their resistance against side-channel attacks. As previously shown in Section 4.5, there are many techniques to satisfy this goal, but most of them unfortunately have a significant impact on the design, disproportionately increasing the resources needed to implement them. The fact of increasing this resistance, but at the same time keeping the algorithm as similar as possible to the original one, is very inviting. For this reason, this technique was chosen to be tested to see if there is actually an improvement by using it. The technique in question is very simple to implement, and is based on mathematical principles. As explained in previous chapters, a cryptographic algorithm, in order to securely encrypt the content of a plaintext, needs confusion and diffusion. To introduce confusion, an S-box is commonly used in cryptographic algorithms, thanks to which the input bits are transformed in a non-linear way, making it difficult for an attacker to trace the key or the original text through cryptanalysis techniques. The S-Box of these two algorithms have been designed in order to have cryptographic properties, such as the nonlinearity and the differential uniformity, optimal against linear and differential analysis, but especially for the case of AES, these same ones are their Achilles heel against side-channel attacks. This is because the designed transformations introduce physical changes observable from the outside that can be exploited by attackers to deduce parts of the secret key, analyzing power consumption or processing time for example. At the beginning of the 2000s, when AES was designed, linear and differential cryptanalysis were the most used types of attacks, as side-channel analysis was still in development, and therefore its creators focused on making this algorithm as robust as possible against these threats. For this reason, side-channel attacks on this algorithm are

surprisingly effective. For ASCON, however, things are different, because it was designed to be resistant against these types of attacks. Nevertheless, this does not mean that it can be improved further. Therefore, the technique that has been used in these two algorithms is to replace the existing S-boxes with variants of them, in order to improve their robustness without going to affect too much their strengths against the other types of cryptoanalysis. To select the alternative S-boxes, the following cryptographic properties were calculated and compared:

- *Nonlinearity (NL)*: this metric has been explained in detail in Chapter 2 in Subsection 2.4.3. For further information, please refer to the above.
- *Differential Uniformity (DU)*: this metric has also been explained in detail in Chapter 2 in Subsection 2.4.4. For further information, please refer to the above.
- *Confusion coefficient variance (CCV)*: in [29] Fei et al. introduces a metric called confusion coefficient, that measures the probability of occurrences for which two key hypoteses k_i and k_j result in different intermediate values v. It can be computed the equation 5.1, where $\mathfrak{L}$ is the leakage function, p is the arbitrary inputs, and $\mathbb{E}$ is the mean operator.

  $$k(k_i, k_j) = \mathbb{E}\left[(\mathfrak{L}(F(k_i \oplus p)) - \mathfrak{L}(F(k_j \oplus p)))^2\right] \tag{5.1}$$

  Then in [30] Picek et al. proposed to calculate the variance of all confusion coefficients. The S-boxes that have a higher value of confusion coefficient variance (CCV) is reported to have better resistance against SCA attacks. This metric is calculated as in equation 5.2.

  $$CCV(F) = var\left(\mathbb{E}\left[(H(F(k_i \oplus p)) - H(F(k_j \oplus p)))^2\right]\right) \tag{5.2}$$

- *Minimum confusion coefficient (MCC)*: it is another important property for SCA. In [31] Guilley et al. explain that when the signal-to-noise-ratio of the leakage is low, the success rate for DPA and CPA mainly depends on the minimum confusion coefficient (MCC). The lower the value of MCC, the lower the success probability to extract the secret key based on leakages associated with the S-box. MCC can be computed as the equation 5.3.

  $$MCC(F) = min_{k \neq k^*}\left(\mathbb{E}\left(\frac{\mathfrak{L}(F(k^* \oplus p)) - \mathfrak{L}(F(k \oplus p))}{2}\right)^2\right) \tag{5.3}$$

- *Transparency order (TO)*: this metric was introduced in 2005 by Prouff in [32] and quantifies the resistance of S-boxes against DPA attacks. This metric

analyzes the correlation between input and output bits with respect to power consumption functions such as Hamming Weight or Hamming Distance. It has been studied that the smaller the transparency order (TO) of an S-box, the higher its resistance would be against this type of attacks. The TO of an (n, m) S-box is defined as the equation 5.4, where D_aF is the derivative of F with respect to a vector a, and W_{D_aF} is the Fourier transform of D_aF.

$$TO(F) = max_{\beta in F_2^m}(|m - 2H(\beta)|- \frac{1}{2^{2n} - 2^n} \sum_{a \in F_2^{n*}} | \sum_{v \in F_2^m, H(v)=1} (-1)^{v \cdot \beta} W_{D_aF}(0, v)|) \tag{5.4}$$

In order to determine what a reasonable value of TO is, Prouff introduced an upper bound and a lower bound for this metric. For a (n, m) function we have the relation 5.5.

$$0 \leq TO \leq m \tag{5.5}$$

In the case of AES, therefore, having a (8,8) S-box, we have that the upper bound is 8, while for ASCON, having a (5,5) S-box, the upper bound is 5. The further this metric is from these values, the better its resistance to this type of attack.

This study also aims to assess how accurately this properties represent the behavior of the S-Box in a practical application. To do this, the results of the attacks obtained are be compared with the calculated metrics to see if the improvement of one of them leads to a comparable improvement in resistance.

The structure of this chapter is organized as follows. In Section 5.2 is presented the hardware setup that was used to collect the power traces useful for performing the power analysis attacks. After that, in Sections 5.3 and 5.4 the countermeasures tested on AES and ASCON respectively are listed. The cryptographic properties calculated for the chosen S-boxes are shown in more detail and can be demonstrated that they show leakage measurable through the TVLA test. The leakage model used to have successful attacks is described, with also the hardware implementation used. Finally the results obtained from the attacks performed is shown. All the conclusions reached are exposed in Chapter 6.

5.2 Experimental Setup

Given the danger of side-channel attacks, especially power analysis attacks, the need arose to make a comparison of the results to understand if an embedded system is more or less secure from this point of view. The purpose of ChipWisperer is precisely this.

ChipWisperer is a side-channel attack platform that includes all the tools needed to perform a side-channel attack: the target device, the measurement equipment, the capture software, and the attack software. This platform makes the study of embedded devices much easier and more cost-effective, and in this way it can help students by making them create a low-cost laboratory, but also researchers because with it they can have an environment that can be replicated all over the world. For this reason, the entire design is open-source (both software and hardware) and can be used with both the supplied target devices, but also with existing hardware, so that if a researcher already has some boards he can continue to use them. In addition, it also includes modules to interface with standard oscilloscopes, encouraging the use of the ChipWhisperer software with existing measurement labs [33].

The unique feature of the ChipWhisperer platform is sample synchronization. Commercial oscilloscopes typically provide their own clock for sampling, which is not synchronized with the device clock, but with the ChipWhisperer-Capture system the clock used for sampling is derived from the device clock, which helps to relax the sample rate requirements, allowing attacks to be made at the same speed as the target device. It is also possible to add an additional delay between the input clock and the sampling point so that can more carefully choose where to sample.

The ChipWisperer platform includes several target boards, which can include various types of devices such as Arm Cortex-M microcontrollers, small FPGAs, PowerPC devices, etc. Thanks to this versatility it is possible to test the algorithms to check their security on different target boards. A really flexible target board is the ChipWisperer CW308 "UFO board" which has many target board possibilities that can fit on the baseboard. The CW305 board is a standalone board that gives the possibility to use a large FPGA, and therefore gives the possibility to test algorithm cores like AES or ECC.

The software used in ChipWisperer is completely written in Python. This makes it very versatile, allowing easy interfacing with other languages such as C/C++ or MATLAB and the possibility of using pre-existing modules useful for cryptography and plotting. The project is divided into two parts: one part is dedicated to capturing and saving power traces, while the second provides all the tools to perform side-channel attacks. Both parts share a common class to manage, save and open projects. The default method used by the software for storage is the Pyhton NumPy library.

In order to perform power analysis attacks the CW305 board, which I mentioned previously, was used in this work. [34]. This board has a USB interface to talk to the FPGA (the Xilinx Artix-7), an external PLL to clock the FPGA, and a programmable VCC-INT power supply. Communication with the Xilinx Artix-7 FPGA is made very simple thanks to the API made available by Chipwisperer

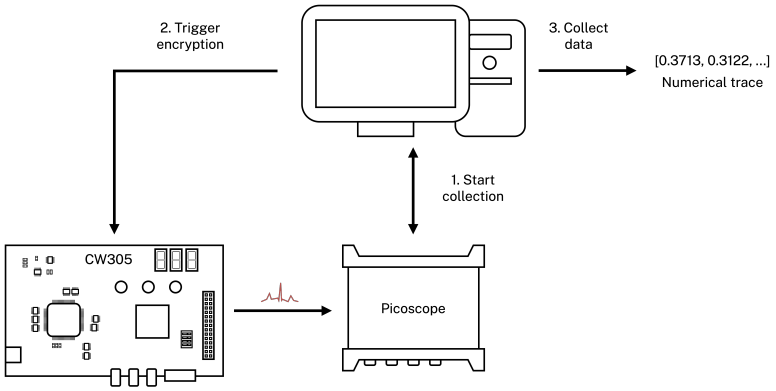


Figure 5.1: Hardware setup to collect power traces.

thanks to which it is possible to write and read on the registers of the target FPGA with the *FPGA_write()* and *FPGA_read()* commands. It also allows to refer to the various addresses not only by address, but also by name, making the code more readable and maintainable. Additionally, a shunt resistor is located between the VCC-INT supply and the FPGA, making it possible to easily perform power measurements useful for side-channel power analysis attacks.

In Figure 5.1 the hardware setup to collect the power traces for this study is shown. First, the oscilloscope, in this case the picoscope 5000A PC based oscilloscope, is armed to start the trace collection. After that, the encryption is triggered on the target device, on which the Verilog code of the encryption algorithm has been loaded. Once the operation is completed, the measured trace is collected from the oscilloscope, and the collection of the next power trace can begin.

5.3 AES lightweight solution

The AES algorithm was chosen among all existing block ciphers for several reasons. The main one is that it is the only existing symmetric encryption standard, and therefore it is the most used cryptographic algorithm in the world, also due to the fact that it is fast, secure, and robust.

In order to achieve the goal of improving an AES implementation against SCA attacks, but without overloading the architecture too much, 8 proposed modifications of the S-box taken from the state of the art were studied.

The first three are taken from [35], where Freyre et al. studied a new hybrid heuristic method based on partitioning the search space in Hamming weight classes, using a new trade-off fitness function. The three S-boxes presented are referred to as *Freyre 1*, *Freyre 2* and *Freyre 3* in this work.

Three other alternatives are taken instead from [36], where an efficient S-box generation technique based on the finite Mordell elliptic curve was developed. They will be referred to as *Azam 1*, *Azam 2*, and *Azam 3*.

The seventh selected S-box is the sixth one proposed in [37] by Hussain et. al, referred to in the original publication as "S-box-6". The methodology used to derive this S-box is based on chaotic systems, trying to improve its cryptanalytic properties, but without taking into account any design criteria against side-channel attacks. It will be referred to as *Hussain 6*.

The last one selected is the one proposed by Ozkaynak in [38], and will be referred to as *Ozkaynak 1*.

5.3.1 Cryptographic properties

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

Table 5.1: Cryptographic properties of the S-boxes in exam for AES. NL: Non-Linearity. DU : Differential Uniformity. CCV : Confusion Coefficient Variance. MCC : Minimum Confusion Coefficient. TO : Transparency Order

Table 5.1 shows the calculated cryptographic properties of the S-boxes under consideration. The first row shows the NL, DU, CCV, MCC, and TO of the standard S-box presented in the original Rijndael version. It can be used to compare other S-boxes under consideration and see if they actually have theoretical improvements regarding cryptanalysis. In fact, for our solutions we need to find the right compromise between improving the resistance against SCA attacks and not worsening too much the robustness against linear and differential cryptanalysis, and this can be seen by making this comparison. For example, we have seen that *nonlinearity* has an upper bound, which makes the S-box more resistant against

linear cryptanalysis, but nonlinearity leads to unpredictable power consumption that can be captured through power analysis. If we compare it with *Azam 2* we can see that the *nonlinearity* has been improved a lot from the point of view of side-channel since we have a decrease in this value, and this leads to a better robustness against CPA attacks, although at the same time there has been a worsening from the point of view of linear cryptanalysis. The same improvement can be noticed by looking at the *minimum confusion coefficient* and the *transparency order*, which even if only slightly, also had a decrease in their values. However, looking at the *differential uniformity* and the *confusion coefficient variance*, as was to be expected, there is a worsening, indicating a lower resistance regarding differential cryptanalysis. Some improvements can be seen even with *Freyre 1*. Unlike *Azam 2* the *NL* does not show such a substantial decrease, but looking at the *CCV* and the *MCC* a substantial increase and a substantial decrease respectively can be noticed, which should lead to a better resistance against SCA attacks. Also in this case, as in the previous one, there is a worsening of the *DU*, but this time instead the increase was smaller, which leads to a worsening with regards to the differential cryptanalysis, but not as critical as for *Azam 2*.

It must always be kept in mind, however, that these are only metrics that indicate the behavior of these S-boxes for this type of cryptanalysis, which may not have such a significant impact on physical applications.

5.3.2 Test Vector Leakage Assessment (TVLA)

Before attempting a power analysis attack, it is necessary to ensure that the implementation has measurable leakage, as a full CPA attack can be very complicated, and the results obtained could be misleading. For this reason, there are many tests that can be performed to see if this requirement is present. One of the most used is the Test Vector Leakage Assessment (TVLA). The basic idea of this test is as follows:

1. Collect two trace datasets G_1 and G_2 according to specific requirements.
2. Divides each dataset in half, so G_1 become G_{1A} and G_{1B}, and G_2 become G_{2A} and G_{2B}.
3. Use Welsh's t-test to test whether G_{1A} and G_{2A} have different means. Perform the same test on G_{1B} and G_{2B} as well.

There are several methods to collect and partition data, the simplest and most powerful one is the Fixed vs. Random Text dataset [39]. To collect the two groups, the following settings for AES-128 are used. For the first dataset G_1, the fixed dataset, I_{fixed} is used as the plaintext and K_{dev} is used for the key.

$$I_{fixed} = 0xda39a3ee5e6b4b0d3255bfef95601890$$
$$K_{dev} = 0x0123456789abcdef123456789abcdef0$$

For the second one G_2, the random dataset, K_{dev} is used as the key, while the plaintext is generated for each power trace by encrypting the plaintext used for the previous trace using K_{gen} as the key.

$$K_{gen} = 0x123456789abcdef123456789abcde0f0$$
$$I_0 = 0x00000000000000000000000000000000$$
$$I_{j+1} = AES(I_j, K_{gen})$$
$$K_{dev} = 0x0123456789abcdef123456789abcdef0$$

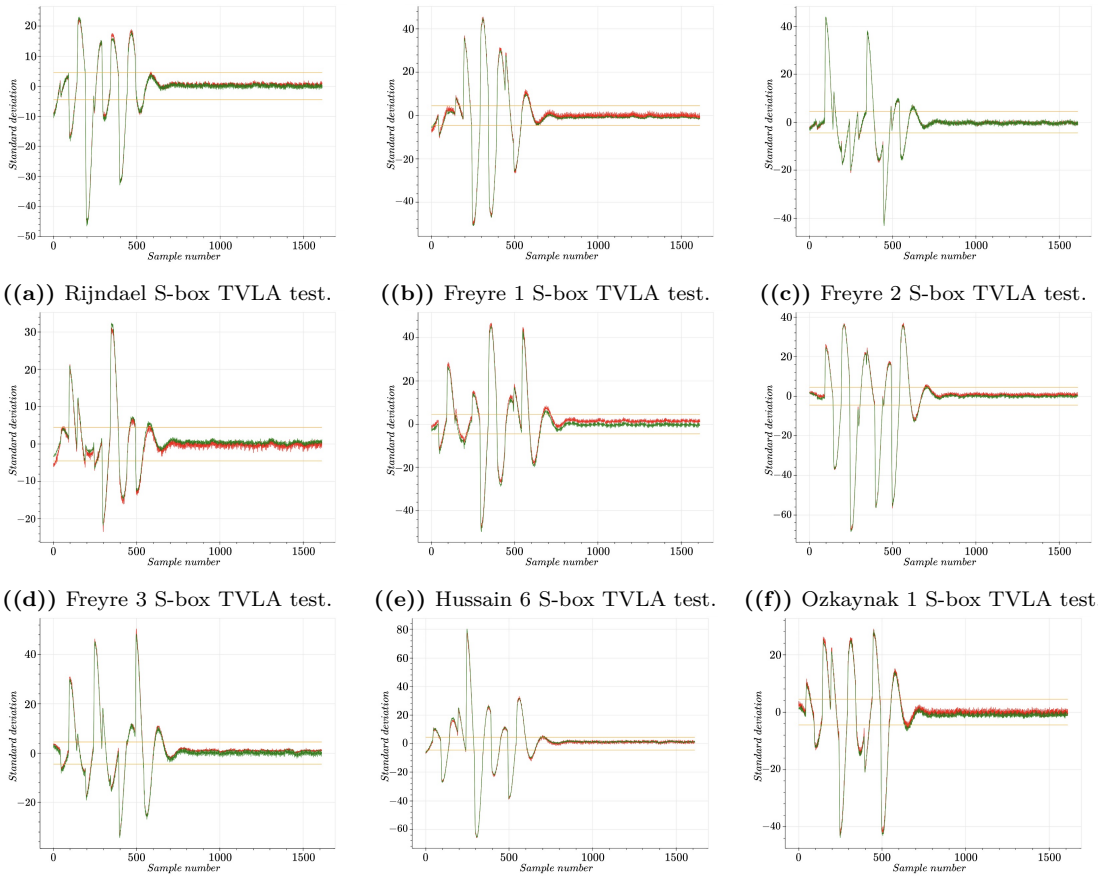
After collecting the two datasets and splitting them in two, the Welsh T-test is performed on them. It is calculated with the equation in 5.6.

$$T = \frac{X_A - X_B}{\sqrt{\frac{S_A^2}{N_A} + \frac{S_B^2}{N_B}}} \tag{5.6}$$

In this equation A and B are the two subsets on which the t-statistic is performed (so G_{1A} and G_{2A} or G_{1B} and G_{2B}), X_A is the mean of all traces in group A, X_B is the mean of all traces in group B, S_A is the sample standard deviation of all traces in group A, S_B is the sample standard deviation of all traces in group B, and N_A and N_B are the number of traces in group A and group B respectively. Note that each trace is a vector of measurements over time, so the means and sample standard deviations are also vectors at the same points in time, and so the t-statistic is calculated point-wise for each time instant in the traces.

Using the Welsh T-test, a threshold of 4.5 is set on the standard deviation, which corresponds to a 99.999% confidence that the difference shown is not due to random change. If both t-statistics calculated for the two subsets exceed $\pm 4.5\sigma$ at the same point in time, then it is considered that the device leak sensitive parameters related to the data.

TVLA is run on all the AES algorithm implementations under investigation to see if there is any measurable leakage. Figure 5.2 shows the results obtained. As can be seen, all the implementations show some leakage in the first part of the graph, because they exceed the orange lines that set the threshold of 4.5 explained earlier, while in the second part it is null. This is because in the first part the execution of the 10 rounds of AES is limited, while in the last part the device is idle, waiting for the next instruction. This result confirms that the device is not secure and shows the points of interest where the CPA attack could be successful. However, it is important to note that the points of interest found may not be



((a)) Rijndael S-box TVLA test. **((b))** Freyre 1 S-box TVLA test. **((c))** Freyre 2 S-box TVLA test.

((d)) Freyre 3 S-box TVLA test. **((e))** Hussain 6 S-box TVLA test. **((f))** Ozkaynak 1 S-box TVLA test.

((g)) Azam 1 S-box TVLA test. **((h))** Azam 2 S-box TVLA test. **((i))** Azam 3 S-box TVLA test.

Figure 5.2: TVLA test of the S-boxes in exam using the Fixed Vs. Random Text dataset. In red the curve for the Fixed Text dataset, in green the one for the Random Text dataset.

indicative, as the power changes may be due to operations that are not exploitable for the attack.

5.3.3 Signal-to-Noise-Ratio and leakage model

While TVLA is a great test for whether there is measurable leakage, it does not provide any information on how that leakage can be exploited. A very useful metric for this purpose is Signal-to-Noise-Ratio (SNR). This is due to the fact that it is linked to mutual information through the observation channel capacity and success rate [40]. In particular, in the AES algorithm, the SNR varies during the rounds due to the progressive diffusion of cryptographic signals and noise. Looking at the structure of this algorithm, one can notice a very important thing: the last round that is executed omits the MixColumns layer. This can be exploited to create an

efficient leakage model, since the absence of this transformation greatly reduces the dispersion of the noise, making the SNR in the last round much higher than the others. If this layer were present, the power would be due to a combination of four different subkeys, thus making the contribution of the attached subkey much lower. Since it is not present, the power signal is more influenced by it instead.

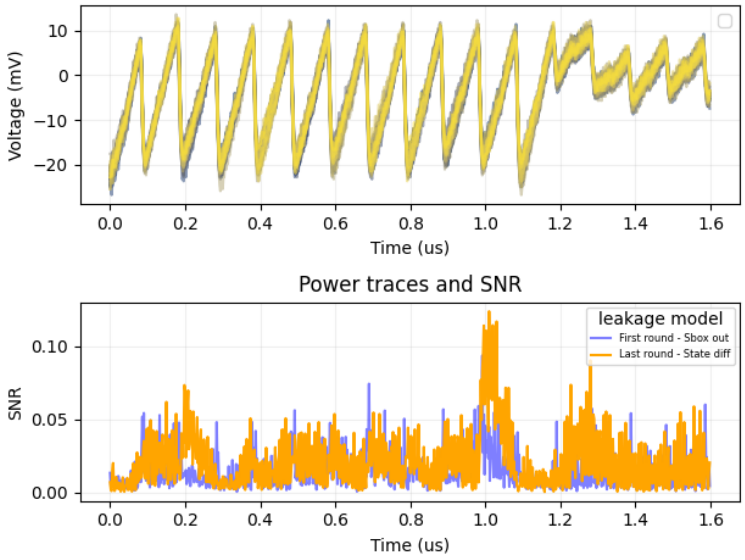


Figure 5.3: Power traces and SNR of two different leakage models.

Figure 5.3 shows this effect. It shows the SNRs obtained with two different leakage models. The first one calculates the leakage of the S-box output in the first round, while the second one calculates the difference of the states between round 9 and round 10. As can be seen, the second leakage model has a higher SNR value than the first one, with a peak during the execution of the last round, thus proving the effectiveness of this choice.

Another benefit of creating a leakage model using the last round of the algorithm is that the omission of the MixColumns layer determines that each byte of the ciphertext is only affected by a single byte of the intermediate value after round 9 and a single byte of the round key. Also, since it is the last round, one can compute the value of the intermediate state after round 9 simply using the obtained ciphertext, without making any complicated assumptions or reconstructions, and with it compute the Hamming distance with respect to the ciphertext to make an assumption about the leakage. All these things lead to a very low level of computational complexity, and therefore it is chosen as the leakage model for CPA attacks in this study.

The verilog implementation of the AES algorithm used executes one round every

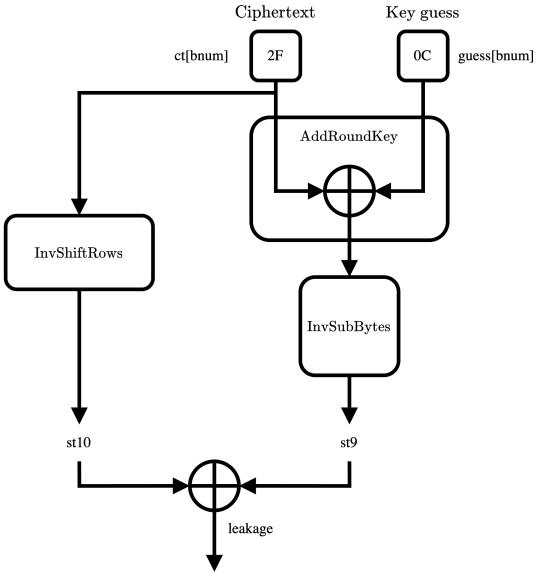


Figure 5.4: Leakage model graphic representation.

clock cycle, and the output of each round is stored in the same set of flops. To determine the leakage hypothesis, the Hamming distance between the contents of the flop after round 9 and the contents of the flop after round 10 is calculated, as shown graphically in Figure 5.4. Note that since in this implementation the ShiftRows operation is executed after round 10, to get the correct content of the flop the ciphertext is manipulated through the InvShiftRows operation to get a correct result.

5.3.4 Success Rate

In order to evaluate the amount of work needed to recover the correct key, the success rate (SR) metric is used. It is part of known-key analysis, i.e. the evaluator knows a priori the key it is trying to recover, and is used to demonstrate the security of a cryptographic implementation [41]. The SR is calculated as follows. Assuming that we are performing a side-channel attack that tries to recover one byte of the AES key, the SR of an experiment i is equal to 1 if the best guess is equal to the correct key, otherwise it is equal to 0.

$$SR^i = \begin{cases} 1, & \text{if } k_c = guess_1 \\ 0, & \text{otherwise} \end{cases}$$

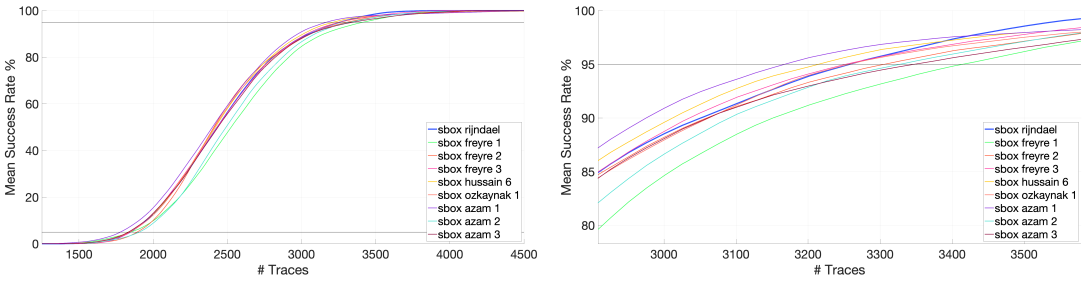
To ensure statistical stability, the calculation of SR^i is typically repeated using

multiple experiments and then averaged, as in equation 5.3.4.

$$SR = \frac{1}{p} \sum_{i=1}^{p} SR^i$$

where p is the total number of experiments that are averaged.

In this study, it is used a modified version of the SR to speed up the computation. The algorithm is considered broken if for 10 consecutive callbacks all bytes of the best guess are equal to those of the correct key. If this condition is met, the SR is set to 1 for all subsequent callbacks and the attack is stopped.



((a)) Mean success rate vs number of traces

((b)) Mean success rate zoom

Figure 5.5: Mean success rate vs. number of traces for the S-boxes in exam.

As previously mentioned, to have a statistical stability this metric is calculated for 200 attacks for each S-box, using 5000 traces for each of them and with a callback every 25 traces. The result obtained is shown in Figure 5.5(a). As can be seen in the graph, the behavior of the implementations with the various S-boxes does not vary much compared to the original *Rijndael* version. However, it should be noted that two of them have a more marked distance than the others, namely *Freyre 1* and *Azam 2*. For these two implementations there is therefore an improvement, although marginal, in the resistance to power analysis.

Table 5.2 shows the number of tracks needed to have an SR of 1%, 5%, 50%, 95% and 99% for a more accurate comparison. Looking at *Freyre 1* and *Azam 2*, the improvement of these two implementations over the *Rijndael* implementation can be immediately observed. What can be also noticed is that, even if the behavior of all the implementations is similar to the reference one, they all have a significant improvement to have an absolute certainty that the attack will succeed. In fact, to have an SR of 99% we have that in the worst case, for *Freyre 3*, the number of tracks increased from 3546 to 3668, that is an increase of 3.44%, while the best improvement is for *Azam 1*, in which the number of tracks increased by 11.98%, from 3546 to 3971.

However, aside from these small improvements, these alternative versions ultimately did not greatly improve the hardware implementation of AES.

S-box type	SR=1%	SR=5%	SR=50%	SR=95%	SR=99%
Rijndael	1646	1844	2432	3259	3546
Freyre 1	1592	1848	2526	3414	3811
Freyre 2	1638	1840	2409	3306	3763
Freyre 3	1696	1845	2432	3252	3668
Hussain 6	1643	1831	2419	3215	3856
Ozkaynak 1	1703	1892	2413	3257	3925
Azam 1	1555	1779	2399	3167	3971
Azam 2	1665	1906	2492	3328	3748
Azam 3	1631	1833	2437	3347	3856

Table 5.2: Number of traces needed to have success rate values of 1%, 5%, 50%, 95%, 99%. The green values highlights a higher number of traces than the Rijndael reference S-box.

5.4 ASCON lightweight solution

ASCON is a fairly recent algorithm, and has become a standard in 2023. Being recently developed, in literature there are not many studies on it, making it the exploration particularly challenging. The alternative S-boxes evaluated as countermeasure in this case are 6. The first one is the one designed by Bilgin et al. in [42], where they propose a new authenticated cipher. In its design, an exhaustive search has been conducted between the quadratic permutations AB and APN in order to be resistant against linear and differential attacks. If these permutations are applied to a compact hardware implementation, such as that of ASCON, they allow to obtain efficiency and resistance against side-channel analysis. For this reason, the 5-bit S-box that is used in this cipher has been chosen, and is referred in this thesis as *lut bilgin*.

The second S-box chosen is the one present in the Shamash cipher [43], proposed by Daniel Penazzi and Miguel Montes and presented to NIST in the competition to become a standard in the field of lightweight cryptography, but which then saw ASCON triumph as the winner. Its unique structure makes it particularly resistant against linear and differential attacks. While the ASCON S-box has 91 linear structures and its inverse includes two undisturbed bits, making it vulnerable in specific rounds, the Shamash S-box contains only 31 linear structures, while its inverse has none, making it particularly difficult for an attacker to find linear patterns, thus increasing its security. This S-box in this works is referred as *lut shamash*.

The last 4 S-boxes used are the ones found by Lu et al. in [44], where they built a model that considers many cryptographic properties simultaneously. First, they transformed the relationship between the S-boxes and their Difference Distribution

Table (DDT) and Linear Approximation Table (LAT) into a satisfiability modulo theories (SMT) problem. Then, they used the requirements on the cryptographic properties as constraints, and with them they used a SMT solver STP (Simple Theorem Prover) to solve the model and find the S-boxes. Using this technique they managed to find several S-boxes that satisfied the requirements. 3 of them were found using properties similar to those of ASCON, such as Differential Uniformity, linearity and the number of BIBO patterns (inputs and outputs with Hamming weights equal to one). These 3 S-boxes are referred to in the following as *lut lu 4*, *lut lu 5*, and *lut lu 6*. For the last one, the model allowed to find a new S-box with improved properties compared to the original ASCON one, and it is referred as *lut lu 7*.

In addition to these 6 S-boxes, the original ASCON S-box is also considered, in both the hardware and lut versions. This is for two main reasons. The first is to have a comparison model with respect to the chosen S-box variants. Furthermore, to investigate whether using the hardware version or the lut version of them gives different results, and, if so, which of the two implementations has better robustness. These two S-boxes are referred to as *hw ascon* for the hardware version and *lut ascon* for the lut version.

5.4.1 Cryptographic properties

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

Table 5.3: Cryptographic properties for the S-boxes in exam for ASCON. NL: nonlinearity. DU: differential uniformity. CCV: confusion coefficient variance. MCC: minimum confusion coefficient. TO: transparency order

Table 5.3 shows the calculated cryptographic properties the S-boxes in question. For the hardware and software versions of ASCON, the first row is used, since they have the same properties and the differences can only be appreciated during attacks.

Going to see the main differences that can be immediately appreciated, for

differential uniformity we have that, as one might expect, for *lut lu 4*, *lut lu 5*, and *lut lu 6* there are no differences compared to the original S-box, this is because they were designed to have cryptographic properties similar to those of ASCON. For *lut lu 7* instead there is a decrease, as well as for *lut bilgin* and *lut shamash* in a more marked way. This leads us to say that their capabilities against differential analysis will be better. In particular we can note that for these last two S-boxes we have a *differential uniformity* value equal to 2, which is the minimum obtainable value and which characterizes them as APN functions, as described in Section 2.4.4.

For the *confusion coefficient variance* things are different. To have a better resistance against SCA attacks this metric must be as high as possible. In this case therefore we have that *lut lu 6* and *lut lu 7* satisfy this requirement. Looking at the *minimum confusion coefficient* instead *lut lu 4* takes the lead having the lowest value for this metric, and therefore a lower probability of extracting the key based on the leakage associated with the S-box. The *transparency order* does not give significant information having all more or less the same value.

Looking at the *nonlinearity* we can notice that all the S-boxes considered have a maximum value equal to the maximum obtainable for this type of functions, as explained in Section 2.4.3, and therefore it cannot be improved further. However, it is found that for *lut bilgin* and *lut shamash* this maximum value is also found for the minimum value of the *nonlinearity*, and this will have to be taken into account when carrying out the attacks.

These results make us understand that it is not possible to improve all the cryptographic properties at the same time. Furthermore, it is not possible to improve cryptanalytic resistance without worsening resistance against side-channel attacks. Therefore, a trade-off must be found between the two, and the purpose of this study is to test whether it is possible to exploit it to obtain satisfactory results. However, we still need to see if this goal has been achieved, and to do this we need to look at their behavior during SCA attacks.

5.4.2 Leakage model and hardware implementation

For ASCON, the initialization phase was chosen as the target for the SCA attack. As previously mentioned, the initial state is formed by the 64-bits initial value (located in register x_0), then there are the 128-bit key (located in registers x_1 and x_2), and finally the 128-bit nonce (located in registers x_3 and x_4). All these components are known, except the value of the key, which is therefore the target for this attack. By hypothesizing the value of the state after the first round of the permutation, and calculating the Hamming Distance between the bits of the state before and after it, it is possible to create the leakage model for this algorithm.

To find which bits of the state registers can be used for the leakage model, the outputs of the state registers can be rewritten in algebraic normal form (ANF) as

the equations from 5.7 to 5.11.

$$y_0 = x_1(x_4 + x_2 + x_0 + 1) + x_3 + x_2 + x_0 \tag{5.7}$$
$$y_1 = (x_3 + 1)(x_2 + x_1) + x_2x_1 + x_4 + x_0 \tag{5.8}$$
$$y_2 = x_4(x_3 + 1) + x_2 + x_1 + 1 \tag{5.9}$$
$$y_3 = (x_0 + 1)(x_4 + x_3) + x_2 + x_1 + x_0 \tag{5.10}$$
$$y_4 = x_1(x_4 + x_0 + 1) + x_3 + x_4 \tag{5.11}$$

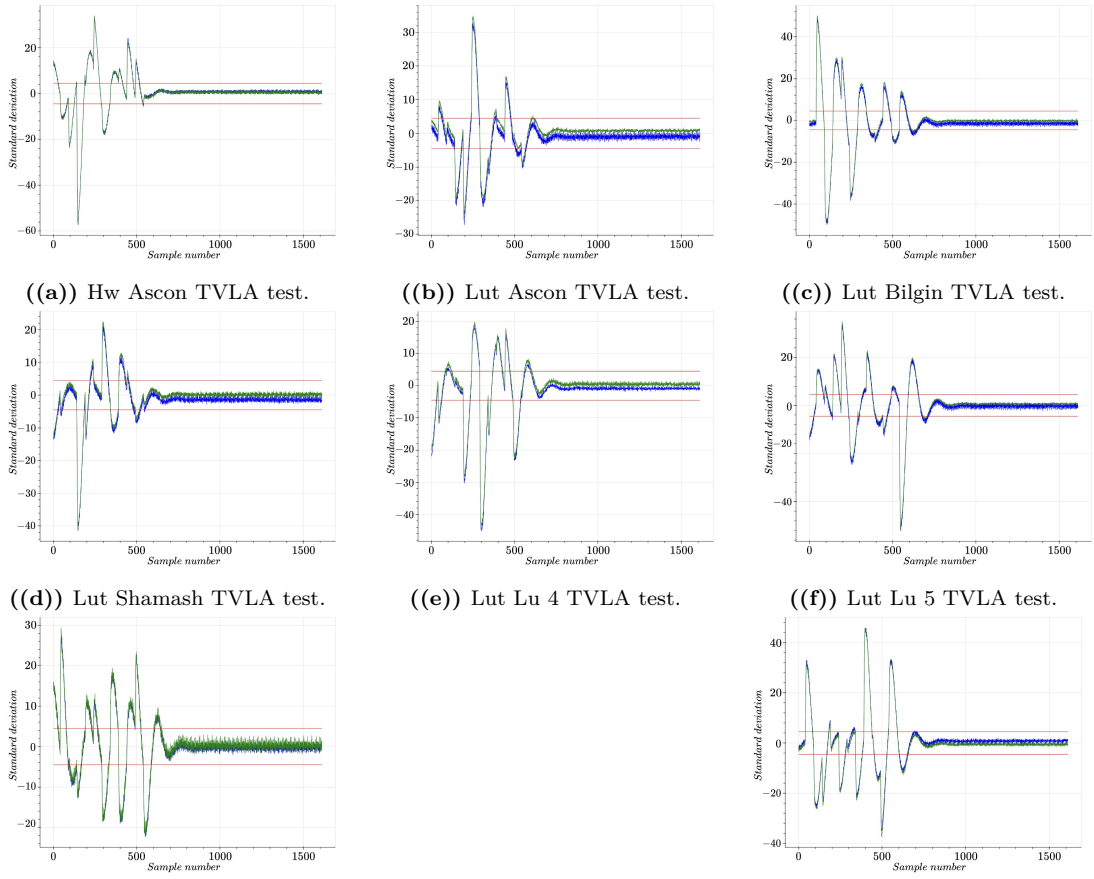
During a CPA attack, a dataset of many power traces is used, all collected with the same key and initial value. The only thing that changes is the nonce value, and so from the previous equations, all constant contributions to power consumption can be removed, namely the terms x_0, x_1, x_2, and all combinations of them. Doing this results in the simplified equations from 5.12 to 5.16.

$$y_0 = x_1(x_4 + 1) + x_3 \tag{5.12}$$
$$y_1 = x_3(x_2 + x_1 + 1) + x_4 \tag{5.13}$$
$$y_2 = x_4(x_3 + 1) + 1 \tag{5.14}$$
$$y_3 = (x_4 + x_3)(x_0 + 1) \tag{5.15}$$
$$y_4 = x_4(x_1 + 1) + x_3 \tag{5.16}$$

As can be seen from these results, equations 5.14 and 5.15 do not include the key bits (which are located in x_1 and x_2) in the calculation, and therefore cannot be used as leakage functions. Equations 5.12 and 5.16 however have a relationship with the bit of x_1, and therefore they can both be used to find the first half of the key that is in this register. To understand which of the two has a better leakage we need to go and see it experimentally. Equation 5.13 instead has leakage on the value $x_1 + x_2$, and therefore once the value of x_1 is found, to find the second half of the key we need to do the XOR between these two values.

Since the attack is entirely based on the initialization phase of the algorithm, which is the first block of operations that is executed in the process, the Verilog code used includes only this step with the 12 permutations of the state registers. Even if the hardware synthesized in this way is much smaller than the one including also the 3 subsequent blocks, this does not affect the power consumption of the implementation, and therefore this choice is made to simplify the study. Furthermore, to increase the trace correlation, the windowing technique for the SCA attack is also used on the collected power trace points, in order to have only those including the first round of permutations, since the analysis includes only that power consumption window.

5.4.3 Test Vector Leakage Assessment



((a)) Hw Ascon TVLA test.

((b)) Lut Ascon TVLA test.

((c)) Lut Bilgin TVLA test.

((d)) Lut Shamash TVLA test.

((e)) Lut Lu 4 TVLA test.

((f)) Lut Lu 5 TVLA test.

((g)) Lut Lu 6 TVLA test.

((h)) Lut Lu 7 TVLA test.

Figure 5.6: TVLA test of the S-boxes in exam using the Fixed Vs. Random Text dataset. In blue the curve for the Fixed Text dataset, in green the one for the Random Text dataset.

Also for ASCON, before moving on to the actual attacks, first is analyzed if there is any measurable leakage in this first phase of the algorithm. As for AES, the TVLA test was used using the Fixed vs. Random dataset. In this case, however, it is no longer the plaintext that varies but the nonce used in this step. The keys used to perform this test are the same as those used for AES and shown in Section 5.3.2, as is the nonce used for the fixed dataset. For the nonces for the random dataset, 128-bit values were generated randomly.

In Figure 5.6 the results obtained by running the TVLA are shown. As can be clearly seen, all the tested implementations have the standard deviation values of both datasets that exceed the threshold of 4.5. This means that also for ASCON the differences that are shown are not due to random changes, and therefore there

is some measurable leakage on sensitive parameters that can be exploited.

With this analysis it is not possible to clearly see which of the implementations has more leakage than the others because all of them have a standard deviation that in absolute value reaches 40. Of particular interest however are *hw ascon*, *lut bilgin* and *lut lu 5* which instead exceed this value reaching up to 50.

5.4.4 CPA analysis

As shown in the previous sections, both registers x_0 and x_4 have a relationship with register x_1, and thus they can both be used to find the first part of the key. Therefore, attacks are first performed on these two registers to see which one performs better. These attacks are also performed using different methods to choose the bits with the best correlation.

In all the attacks is performed a preprocessing to improve the calculated correlation level. The shape of the correlation curve over time is more informative than its absolute value itself, because prior research, such as [45], has demostrated that *"when correlated working variable causes a change in the power consumption of the device, an edge typically appears in the correlation trace"*. Since the edge detection operators are very sensitive to noise, a Gaussian filter is used to filter it.

To perform the edge detection instead, the first derivative of the correlation traces is calculated, and the points in which the derivative reaches the maximum are identified as potential edges. In this way, an alternative distinguisher to the Pearson correlation coefficient is used to select the subkeys. By shifting the focus to significant changes in correlation, rather than looking at its value, this technique becomes more robust in noisy environments, thus leading to better results with fewer traces. Algorithm 8 shows the pseudocode of how this technique is applied.

Algorithm 8 Pseudocode of the alternative distinguisher to the Pearson correlation coefficient.

Require: *cpaout*: correlation of the subkey
Ensure: *maxcpa*: maximum value of the derivative
1: ▷ **Step 1: Smooth the correlation trace**
2: *cpaout* ← Gaussian_filter(*cpaout*)
3: ▷ **Step 2: Compute the first derivative**
4: *cpaout* ← First_derivative(*cpaout*)
5: ▷ **Return the value of the edge detection**
6: *maxcpa* ← *max*(*abs*(*cpaout*))

Since the subkeys found in the attack are composed of three scattered bits of the key due to the linear diffusion layer, to build the complete key it is necessary to select the bit with the highest correlation.

Example. To give a better understanding, bit 0 of register x_1 by attacking register x_0 can be found by attacking column 0, but that same bit is also involved by attacking column 36 and column 45 due to the right shift of the linear diffusion layer.

Since each bit of the key can be found by attacking 3 different columns, you have to choose which of the 3 has a higher probability of recovering the correct value of the bit. This could be done simply by going to see which one has the highest value of the correlation, but for a more in-depth analysis, different methodologies have been studied that also look at other aspects in addition to that.

The first method uses 3 different metrics: the first metric is the highest correlation level, the second metric is the greatest distance between the first subkey and the second subkey with the highest correlation level, and the third is the multiplication between these two values. Going to see the maximum value that is obtained in these metrics, you can obtain 3 possible values that the bit can assume. Then going to choose by majority between these 3 results, you obtain the hypothetical value of the key bit that has the highest probability of being correct. The pseudocode of how this choice is made is shown in Algorithm 9.

The second method calculates the leakage that can be had with a certain column by doing the TVLA test on the correlation level, and then only the columns with a higher level of leakage are chosen to make the attack.

The last method is instead a combination of the two: the first two metrics that are calculated are the correlation and the distance, like the first method, while the third metric is the amount of leakage of a certain column, like the second method. Then, like the first method, the choice is made by majority among the 3 bits found.

Figure 5.7 shows the results obtained by performing the attacks with these methods on the *hw ascon* implementation. To understand which method is better, and subsequently which implementation is better, we looked at the number of correct bits found as the number of power traces processed increased. As can be seen, the second method presented, that is, choosing the columns with the highest leakage and then running the attack only on them, is the one that has a worse performance. This method was executed in two variations where the only thing that changes is the number of traces used to calculate the measurable leakage, in this case 20k and 40k traces. With these two methods, it was possible to reach a maximum of 50 correct bits with 60k traces.

What instead seems to be the best register with which to perform the attack is the register x_4. In fact, can be seen how, both using the first method (the red curve) and the third method (the fuchsia curve), all 64 bits of the first part of the key

Algorithm 9 Bit selection pseudocode by attacking register x_4.

Initialize max_corr with the correlation level of the first bit
Initialize max_diff_corr with the difference of the correlation levels of the first bit
Initialize max_mul with the multiplication between the correlation level and the difference of the correlation level of the first bit
Initialize max_bit_0, max_bit_1, and max_bit_2 with the value of the first bit
if $max\_corr <$ correlation level of the second bit **then**
    Update $max\_corr \leftarrow$ correlation level of the second bit
    Update $max\_bit\_0 \leftarrow$ value of the second bit
end if
if $max\_diff\_corr <$ difference of the correlation levels of the second bit **then**
    Update $max\_diff\_corr \leftarrow$ difference of the correlation levels of the second bit
    Update $max\_bit\_1 \leftarrow$ value of the second bit
end if
if $max\_mul <$ multiplication between the correlation level and the difference of the correlation level of the second bit **then**
    Update $max\_mul \leftarrow$ multiplication between the correlation level and the difference of the correlation level of the second bit
    Update $max\_bit\_2 \leftarrow$ value of the second bit
end if
if $max\_corr <$ correlation level of the third bit **then**
    Update $max\_bit\_0 \leftarrow$ value of the third bit
end if
if $max\_diff\_corr <$ difference of the correlation levels of the third bit **then**
    Update $max\_bit\_1 \leftarrow$ value of the third bit
end if
if $max\_mul <$ multiplication between the correlation level and the difference of the correlation level of the third bit **then**
    Update $max\_bit\_2 \leftarrow$ value of the third bit
end if
if $max\_bit\_0 == max\_bit\_1$ **then**
    Assign $guess\_bit \leftarrow max\_bit\_0$.
else if $max\_bit\_0 == max\_bit\_2$ **then**
    Assign $guess\_bit \leftarrow max\_bit\_0$.
else
    Assign $guess\_bit \leftarrow max\_bit\_1$.
end if

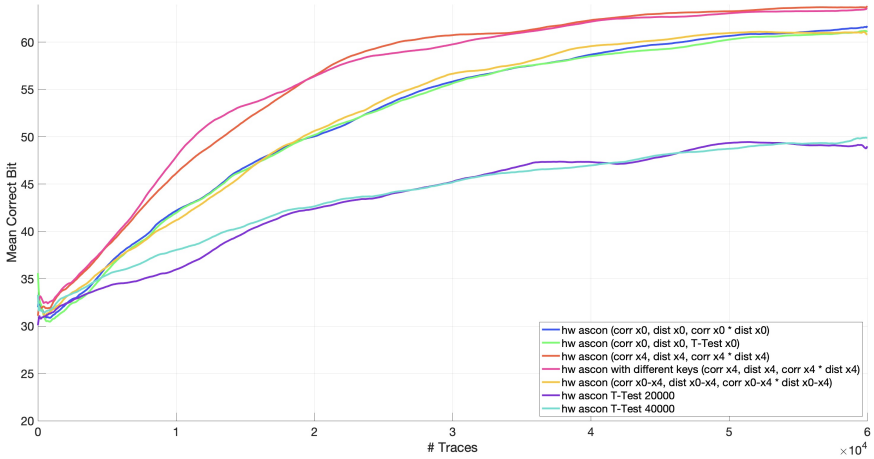
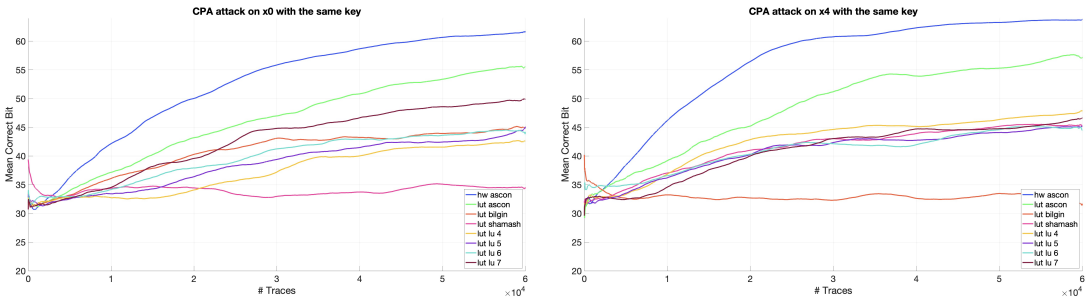


Figure 5.7: Comparison between methods to attack ASCON.

contained in the register x_1 can be found. The register x_0 also performs well, but it cannot completely recover the content of the register x_1 with this number of traces. Given the good performance of these two registers, we thought of performing an attack using both of them, and therefore making the choice between 6 bits, but as can be seen, the curve of this attack (the yellow curve) has a performance more similar to the one done with x_0.



((a)) CPA on register x0.

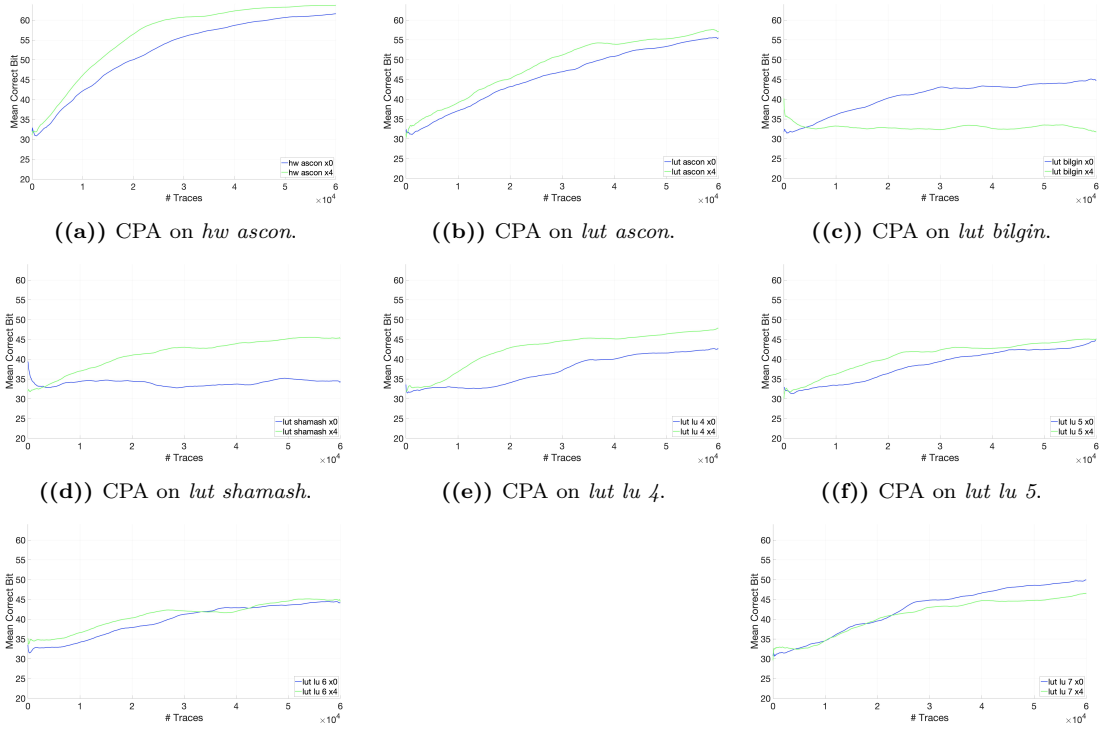
((b)) CPA on register x4.

Figure 5.8: CPA attack on registers x0 and x4 for the S-boxes in exam.

Given the good performance of the first bit selection method, it was selected to perform the subsequent attacks. The next step is to attack the registers x_0 and x_4 to see the behavior of the selected S-boxes. In Figure 5.8 the obtained results are shown. As can be seen, the worst implementation among all is *hw ascon*. In fact, in both attacks it is the implementation with the highest number of correct bits found. This implementation is immediately followed by *lut ascon*, which is the

same S-box implemented in the form of a look-up table. Below them, instead, we find all the other S-boxes under examination. As can be seen, in this case, unlike AES, we have a notable improvement in terms of robustness against power analysis, because even with 60k traces it is not possible to obtain more than 50 correct bits found.

A very interesting thing can be noticed in Figure 5.9, where the curves of each S-box are compared for the attacks on x_0 and x_4. As can be seen, the attack on x_4 is the one that has a better performance for almost all implementations. Going into more detail, however, it can be noted that *lut bilgin* and *lut shamash* have particular behaviors. The first one during the attack on x_4 seems to have some random leakage on this register, and therefore on average it is possible to find 32 correct bits. The second S-box instead seems to have the same behavior but with the attack on x_0. These behaviors should be investigated further to understand if they have perfect properties against this type of cryptanalysis.



((a)) CPA on *hw ascon*. **((b))** CPA on *lut ascon*. **((c))** CPA on *lut bilgin*.

((d)) CPA on *lut shamash*. **((e))** CPA on *lut lu 4*. **((f))** CPA on *lut lu 5*.

((g)) CPA on *lut lu 6*. **((h))** CPA on *lut lu 7*.

Figure 5.9: Comparison of attacks on the x0 and x4 registers for the different S-boxes under consideration.

Thus far, all attacks have been executed using the same key. This raises the possibility that the results may not be generalizable. To address this, we tested whether changing the keys for each attack would produce different results. To do

this, the register with the best performance, and therefore x_4, was chosen as the register for the attack, and a different random key was generated for each dataset of traces. Figure 5.10 shows the results obtained. Even for these attacks it can be seen how the chosen S-boxes have a very high resistance against CPA attacks, since the number of correct bits with 60k traces is much lower than those found for the two versions of ASCON. This test is also very interesting because it shows how the key used to perform the attacks gives better results than the case of a random key, further confirming the robustness of these implementations.

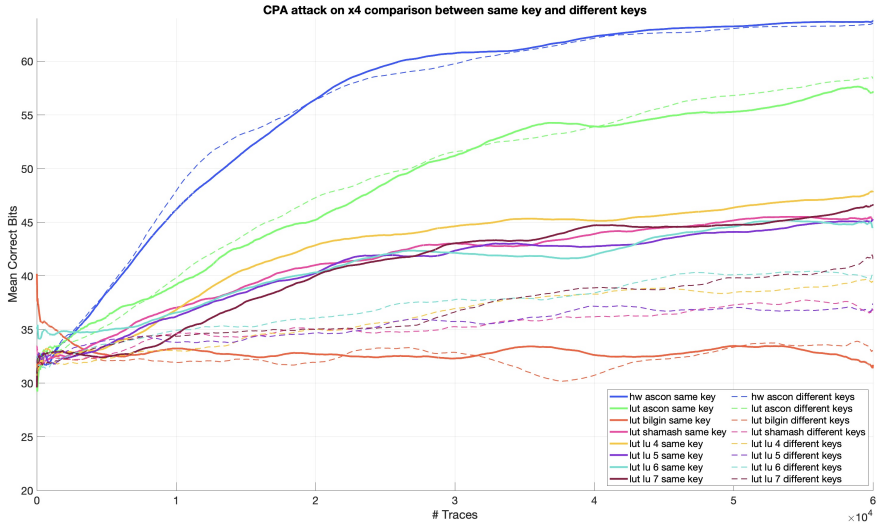
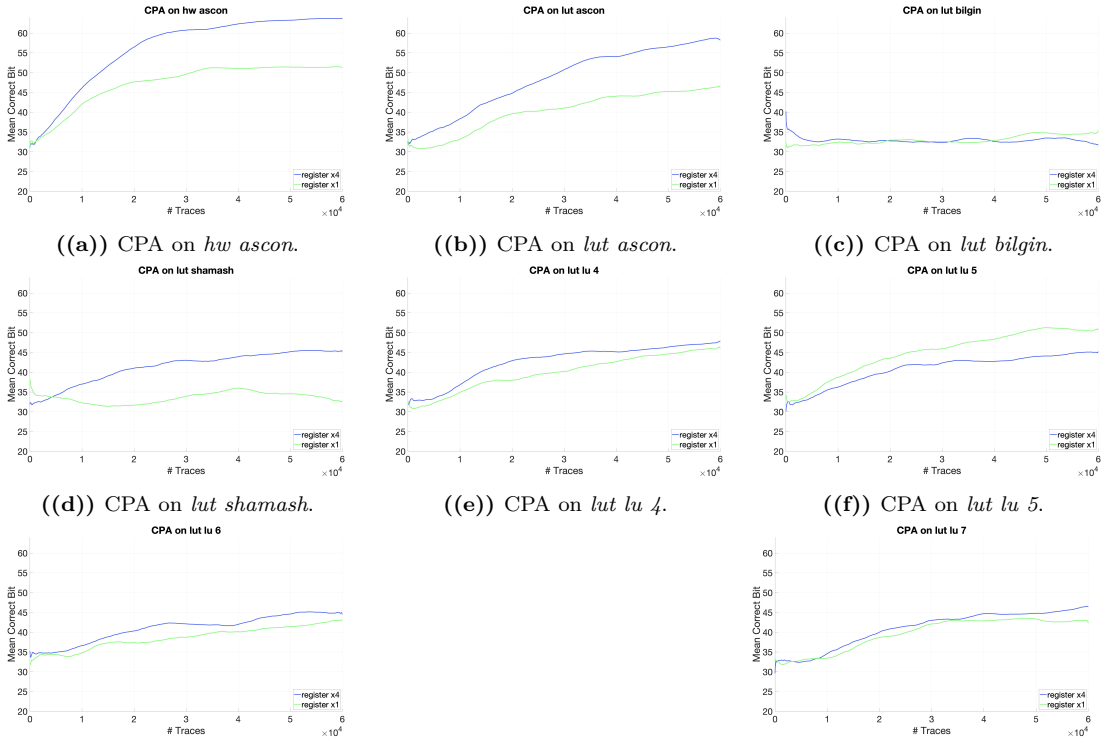


Figure 5.10: Comparison of CPA attack on x4 using traces collected with the same key and with different keys.

Once finished the attack on the first part of the key, the attack can then proceed on to the final phase, that is, attacking the register x_1 which has a relation with $x_1 + x_2$. Therefore, after finding the first part of the key, the second part of the key can be recovered by doing an XOR between the bits found with the attack and the bits of the register x_1, thus discovering the bits of the register x_2. Since attacking register x_4 gave better results than attacking x_0 when it comes to recovering the first part of the key, this register is used to recover x_1. Figure 5.11 shows the results obtained by the attacks on the S-boxes under consideration for the attack of x_4 and x_1.

As can be seen from the curves, register x_4 has much more leakage than register x_1, and therefore with the number of traces used to launch this attack, even for the weakest S-box, i.e. *hw ascon*, it is not possible to obtain all 128 bits of the key. In particular, there are several interesting details. For *hw ascon* we have that after 30k traces it seems that the number of correct bits found settles around 52 bits.

For *lut bilgin* instead, having this random behavior and therefore not being able to find the correct bits of the key, even the attack on x_1 has this behavior, making it so far the best S-box to use to increase robustness. Another interesting thing can be noted for *lut shamash*. This S-box has a behavior similar to *lut bilgin* when the register x_0 is used as a leakage model. This same behavior seems to be present also when the register x_1 is used, in fact as can be noticed in Figure 5.11(d) the number of correct bits found is always between 30 and 35.



((a)) CPA on *hw ascon*. **((b))** CPA on *lut ascon*. **((c))** CPA on *lut bilgin*.

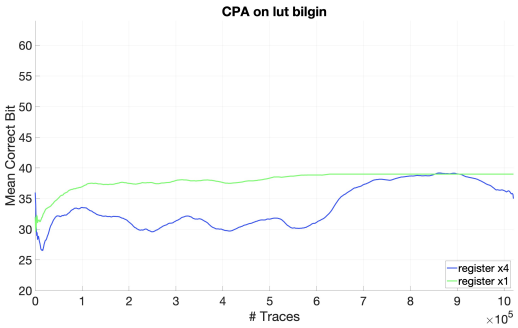
((d)) CPA on *lut shamash*. **((e))** CPA on *lut lu 4*. **((f))** CPA on *lut lu 5*.

((g)) CPA on *lut lu 6*. **((h))** CPA on *lut lu 7*.

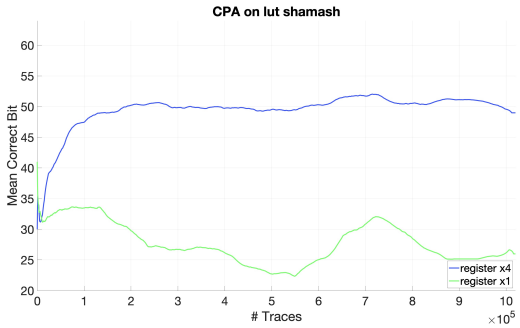
Figure 5.11: CPA attack on both x_1 register and x_2 register to recover the complete key for the different S-boxes under consideration.

In Figure 5.10 it can be seen that using different keys, the 3 S-boxes that have a better behavior regarding side-channel analysis are *lut bilgin*, *lut shamash* and *lut lu 5*. Since the attacks for these analyses were done on 60k traces, to have a more complete view of their behavior, an attack using 1M traces was performed, and the result of this attack is shown in Figure 5.12. To perform this analysis, both the x_4 and x_1 registers were attacked, in order to have an even more complete view of their behavior. The first interesting thing that can be noticed is that all three attacks have the same pattern: the number of correct bits that can be found increases with the increase in the number of traces and then from about 200k traces onwards it stabilizes at a fixed value. This makes it clear that with these implementations

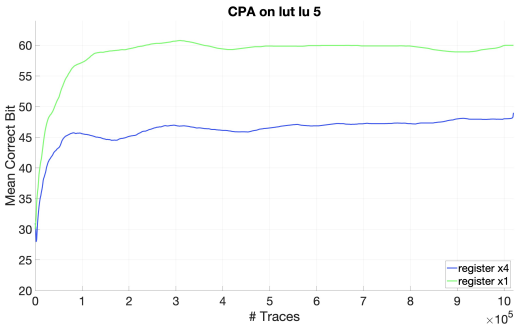
some bits have leakage, while others do not. The other interesting thing is that not all implementations have the same number of bits with leakage.



((a)) CPA on *lut bilgin* with 1M traces.



((b)) CPA on *lut shamash* with 1M traces.



((c)) CPA on *lut lu 5* with 1M traces.

Figure 5.12: CPA attack with 1M traces on the 3 best S-boxes that were obtained.

As expected, the behavior of *lut bilgin* remains rather unchanged compared to the case with 60k tracks, reaching a maximum of less than 40 correct bits found (Figure 5.12(a)), and thus confirming itself as the best S-box among all those examined. The interesting thing that this attack gives us, however, is that it makes us understand that the correct bits that have been found are not due to a random select of bits, and therefore since the value that they can assume is only 1 or 0, statistically we would have half of the correct bits. Instead, as noticeable by the trend of the curve, there is an increase, albeit minimal, in the number of bits found, and then a stabilization, and therefore this makes us understand the correctness of the attack that was made, and therefore their very small number is given only by the good cryptographic properties of the S-box.

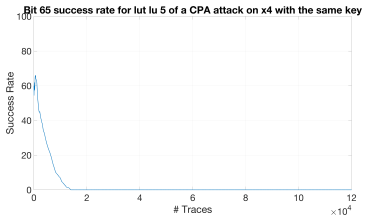
Lut bilgin, together with *lut lu 5*, also have another strange behavior. As explained in Subsection5.4.2, to find the value of the bits of the second half of the key you need to know the value of the bits for the first half of it, because x_1 has leakage on $x_1 + x_2$. In these two implementations, however, the number of correct bits for the second half of the key is greater than the first. This makes us

understand that the register x_1 has a greater leakage than x_4, and and hence it is able to correctly recover the bits of x_2 even with bits of x_1 that are incorrect.

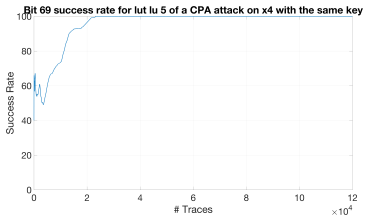
The last thing that is brought to light with this attack is the fact that for *lut shamash*, as can be seen in Figure 5.12(b), in this case the register x_1 has much less leakage than x_4, with a very low average of bits found, around 25 correct bits. This gives us an idea of how robust *lut shamash* and *lut bilgin* are against side-channel analysis.

The fact that all implementations have a limit of correct bits found beyond which they cannot go raises the question whether there are specific bits that have a lot of leakage and can therefore always be found, or, more importantly, whether there are bits with such low leakage that they can never be recovered. In this regard, a final analysis was made on these three S-boxes, calculating the success rate for each bit of the key. This analysis was done on 5 attacks with 120k traces, and very interesting results were obtained.

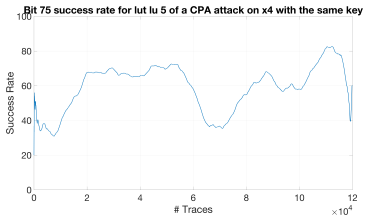
By attacking *lut lu 5* the success rates obtained for all 64 bits of the key can be grouped into 3 categories, of which 3 examples can be seen in Figure 5.13. The first group consists of bits that can never be recovered, and an example is the graph of bit 65 in Figure 5.13(a). This group includes 12 bits of the key, namely bits 64, 65, 66, 73, 74, 80, 82, 90, 96, 97, 105, and 112. As can be seen in the figure, the success rate starts from 50%, being the beginning and therefore the bit has a random value, but as the correlation value is calculated with an increasing number of traces it decreases until it reaches a success rate of 0%. These bits therefore seem to have such a low leakage as to make their recovery impossible.



((a)) Success rate of bit 65 attacking x_4 of *lut lu 5*.



((b)) Success rate of bit 69 attacking x_4 of *lut lu 5*.



((c)) Success rate of bit 75 attacking x_4 of *lut lu 5*.

Figure 5.13: Success rate of bits of *lut lu 5* by attacking x_4.

The second group is made up of those bits that have such a high leakage that they can always be recovered. An example is bit 69, whose graph is shown in Figure 5.13(b). This group includes bits 69, 70, 71, 72, 76, 77, 78, 79, 84, 85, 86, 87, 88, 89, 92, 94, 95, 98, 100, 101, 102, 103, 104, 106, 108, 110, 111, 113, 114, 118, 119, 120, 121, 122, 125, 126, and 127. As you can see, their behavior is very similar to that of the bits that are never recovered, only this time the success rate increases, instead of decreasing, and reaches a the value of 100%.

The remaining bits instead are part of the last group, and have a behavior like the one in Figure 5.13(c). As you can see, in this case on average they are recovered half the time, so they have some leakage but only for certain input configurations.

Attacking *lut shamash* instead yields a more surprising result. Since the attack result on x_4 is better than the attacks on the same register for the other two S-boxes, as can be seen in Figure 5.12, there are no bits that are never recovered. However, the bits that are always recovered are fewer than in the previous case, 31 versus 37.

For the last S-box, *lut bilgin*, we have only one bit that is always recovered, bit 64. All the other bits have a fairly random behavior, with an average recovery of 50% in this case too. This behavior is highlighted by the fact that it is never possible to recover more than 40 bits with the attack. This is the strong point of this implementation, which makes it so robust against side-channel attacks.

These results are surprising, but they are still only preliminary. Since the success rate calculation was done only on 5 attacks, it is not possible to fully see the statistical behavior of these implementations. To have more correct results, this type of analysis should be continued by integrating the results obtained with a greater number of attacks, so as to completely eliminate the influence of the single attack and bring out their behavior completely.

Chapter 6

Conclusions

The results obtained and presented in Chapter 5 have given very clear answers regarding the strength of this lightweight countermeasure against side-channel attacks. In the case of AES, the use or not of an S-box different from the original implementation does not imply substantial changes. As seen in Subsection 5.3.4, all the alternative versions of the S-boxes used have the same behavior as the version designed by Rijmen and Daemen. The main innovation, however, that led to the study of this algorithm is the leakage model used. As has been seen, the last round of AES is the most suitable to attack in the case of hardware implementations, since the SNR is very high in this block due to the lack of the MixColumns layer. Using this leakage model therefore, in 5 minutes and with less than 4000 traces it is possible to completely crack AES.

In the case of ASCON, however, the situation is different. It has been widely demonstrated with all the analyses performed previously that this countermeasure is very effective for this algorithm. ASCON was designed from the very beginning to be robust against side-channel attacks, as evidenced by the fact that breaking the algorithm using the hardware implementation of its S-box requires 60,000 traces — significantly more than what is needed for AES. This resistance in ASCON is due to the fact that only 2 bits of the key enter its substitution layer, which is the component that consumes the most power, and therefore the leakage is significantly decreased compared to the case of AES where the bits of the key that enter the S-box are 8. However, using different versions of the S-box, with improved side-channel resistance, brings this resistance to very high levels, as in the case of the S-box *lut bilgin*. Using this S-box in particular, with 1M traces this type of analysis is performed in a time 343 times greater than that required to violate the AES algorithm, without however managing to find more than 80 correct bits of the key. This confirms the success of this study regarding the ASCON algorithm.

For ASCON, however, the things that this study has brought to light are not over. Being a very recently standardized algorithm, in the literature there are not many

in-depth studies regarding side-channel attacks on its hardware implementations. Therefore, in order to arrive at an attack that could work on it, it was necessary to investigate which is the most suitable leakage model. In this case, it was studied that the initialization step of the algorithm has measurable leakage that can be exploited. In particular, there are different registers that can be attacked, each with a different level of leakage. In this study, it was demonstrated that by attacking register x_0 and register x_4 it is possible to recover the contents of register x_1, where the first half of the key is present. By attacking register x_1 instead it is possible to recover the contents of x_2, where the second one is present.

The subsection 5.4.4 presents the analysis performed to determine which of the two registers between x_0 and x_4 has the best leakage, and to do this different techniques are tested to find the one with the best performance. Since a certain bit of the key can be recovered by attacking 3 different columns, it is necessary to choose those with the highest leakage in order to minimize the number of necessary traces, and therefore minimize the time needed to execute the attack. The worst technique to make this kind of choice is the one where a preliminary study of the leakage is done for each column, and then the attack is performed only on those with the highest leakage. Although there is a halving of the execution time of the complete attack on the same number of traces, the number of correct bits found still fails to exceed 50. The experimental results show that the best solution among those tested is to determine by majority the value of the bit based on three metrics: the correlation value, the distance between the subkey with the highest correlation level and the second one, and the multiplication between these two metrics. With this technique, attacking register x_4, it is possible to break all 64 bits of the *hw ascon* implementation with the number of traces under consideration. Although register x_0 also has good results using this method, it is only possible to find 60 correct bits of the key on average, thus confirming register x_4 as the one with the highest leakage of the two.

To obtain these good results, however, it is necessary to do a preprocessing on the correlation levels of the subkeys found, in order to improve the results obtained. In fact, it has been studied that the shape of the correlation curve is much more informative than its value itself. Of particular interest are the edges in the correlation trace, which indicate that a correlated working variable has caused a change in the power consumption of the device. For this purpose, an alternative distinguisher is used instead of the Pearson correlation coefficient, in which in a first step the noise is filtered from the correlation traces using a Gaussian filter, after which the first derivative of the trace is made and the maximum absolute value of the result is taken, which identifies the potential edges.

Testing whether this lightweight countermeasure is valid is not the only purpose of this study, however. Another question that needs to be answered is whether the S-box's possession of good cryptographic properties has any effect in practical

application. In the case of AES, since the countermeasure is little or not at all effective, this comparison is difficult to make, and one might come to the conclusion that the cryptographic properties used do not play a fundamental role in describing the behavior of a function. For this algorithm, it has been shown that the implementations with a slightly better performance than the others are *Freyre 1* and *Azam 2*. For the first of the two, looking at its cryptographic properties in Table 5.1, one can see that for nonlinearity and differential uniformity there is no improvement. Nonlinearity has an upper bound of 116,686, and the closer its value for an S-box is to this bound, the more resistant it is to linear cryptanalysis. However, nonlinearity is a big problem for side-channel analysis as it leads to unexpected energy consumption that can be brought to light by these types of attacks. To be more resistant to power analysis, this property must be minimized, although in this way linear cryptanalysis is compromised. For differential uniformity instead, there is a lower limit of 2 that can be reached, which identifies a function from being an Almost Perfect Linear function. However, it has been seen that for AES the best value that can assume is 4, thus having a good compromise between resistance to attacks and practical implementation. In the case of *Freyre 1* these two properties have the value of 110 and 8 respectively. For nonlinearity there is a slight decrease compared to the original S-box (112), while differential uniformity instead undergoes a large worsening, both compared to the original implementation (4), but also from the point of view of the limit that can be obtained. For the last three metrics, however, there is a great improvement. The confusion coefficient variance must be as high as possible to improve the resistance against SCA attacks, and this S-box gets a value of 4.5, much higher than Rinjdael which has 0.11. To have a lower probability of success in extracting the secret key based on the leakage associated with the S-box, the minimum confusion coefficient must be as low as possible, and also in this case it has a value of 0.13, 6 times lower than the original version (0.81). Finally, the transparency order has an upper limit of 8 in the case of AES, and the more this metric moves away from this value, the more the associated S-box is resistant to power analysis. Since for *Freyre 1* this value is the lowest compared to all the other S-boxes considered, this property may also have had an influence on the performance of this implementation. One thing that can be noticed by looking at that table is that *Freye 2* also has the same values for these cryptographic properties, but compared to *Freyre 1* it did not have the expected results. This point provides us with a very important piece of information: although these metrics correctly describe the behavior of some S-boxes, they are only partial and not sufficient to evaluate the implementation resistance of all of them.

For *Azam 2* instead we have the opposite situation. We have that the CCV, the MCC, the TO and the DU are much worse, both compared to the original S-box, but above all with regards to the value that they should assume to have a

good resistance against side-channel attacks. In this case however we have that the minimum nonlinearity reaches a value of 90, much lower than Rijndael and also excellent for showing excellent resistance to side-channel attacks. This fact could give us a clue as to the weight that these metrics have on the behavior of the implementations. In fact, going to look at the nonlinearity in the other implementations, we can notice that *Ozkaynak 1* has the minimum among all the S-boxes of the minimum nonlinearity, and although its behavior during CPA attacks is not very different from that of Rinjdael, it can be seen in Table 5.2 that it nevertheless reaches a success rate of 99% for a number of traces among the highest.

More significant is the case of ASCON. For this algorithm, the S-boxes with the best performance were found to be *lut bilgin* and *lut shamash*, having almost zero leakage for certain registers used in the attacks. The first thing that catches the eye when looking at the Table 5.3 that shows their cryptographic properties is the fact that for both of these implementations there is a perfect value for the DU, that is 2. This metric could be the fundamental factor for their great robustness against side-channel attacks. In fact, looking at all the other properties taken into consideration, it can be noted that there are no other improvements compared to the original version of ASCON, indeed all the other metrics have worsened a lot, moving away from their optimal value. Also in this case, therefore, we can come to the conclusion that differential uniformity plays a fundamental role in the description of the S-boxes.

As for the other implementations, their robustness can be found in the other properties. All of them have a minimum nonlinearity of 8, which is excellent for improving resistance to power analysis. In addition, there is also an improvement for *Lut Lu 4*, *Lut Lu 6* and *Lut Lu 7* regarding the CCV, which increases compared to the original case. For *Lut Lu 4* we have a further improvement in the MCC, which has the minimum value among all the other implementations. For *Lut Lu 5*, however, there are no particular further improvements, so even in this case it is necessary to investigate whether other cryptographic properties exist to describe its excellent robustness to side-channel analysis.

Cryptographic properties and attack results can be further compared to get additional considerations. Making a numerical comparison of the improvements, we can notice a very important thing: if we take the CCV for ASCON and we go to see how much it has been improved for the various alternative S-boxes we can see that for example, in the case of *Lut Lu 6*, it is 1.77 times larger than the original version. So we would expect that the number of traces needed to get the same result for this implementation would be 1.77 times larger than the *Hw Ascon* implementation. In this case instead we had a 6.66× increase in the number of traces needed to discover 37 bits of the key in both cases. Making the exact same type of consideration for *Lut Lu 4* instead, on the one hand we

have a 1.16× increase in cryptographic property, while the number of traces has increased by 7.50×. This makes us understand that there is no direct relationship between cryptographic properties and the improvement of resistance in practical application, which is further proven by the fact that even in the case in which there is no such improvement on theoretical data, in practice a greater robustness has been found. These considerations can be applied to all the cryptographic properties considered, and this leads to a very important conclusion. Although these metrics can be used preliminarily to have a general idea of the behavior of an S-box, they cannot be used alone to intuit the resistance of an implementation. To have a correct intuition one must consider them as a whole, and many more metrics than those considered in this thesis are needed to better understand their behavior. Furthermore, the results obtained in the theory cannot be used as proof of the cryptographic resistance of an implementation, because there is no mathematical relationship between the improvement of a metric and the increase in the number of required traces. Therefore, cryptographic properties must always be supported by experimental results.

In conclusion, in light of the results obtained, we can give an overall judgment on the effectiveness of this lightweight countermeasure to increase resistance to side-channel attacks. By increasing this resistance, we lose something in terms of robustness to cryptanalysis, so we need to understand if this trade-off makes sense to do, or if it is better to use these two algorithms as they were designed. In the case of ASCON, it is easy to give an answer. The attacks that have been carried out have all shown how this countermeasure is very effective, greatly increasing the number of traces needed for the attack, and even leading to not being able to find all the bits needed to discover the secret key. In this case, this trade-off has more positive aspects than negative ones, and therefore it is very advantageous to do it. In the case of AES, however, the results obtained are not as hoped. In ASCON the S-box plays a fundamental role in data confusion and diffusion, being an algorithm with a focus on efficiency and simplicity, but for AES instead its multi-layered structure, being more complex, already provides a high level of confusion and diffusion, so the role of the S-box in security is much more marginal. Although there has been a slight improvement, as a result this trade-off is not very convenient.

Given the limited time available for writing this thesis, and given the fact that an attack on ASCON requires a fair amount of time to be executed, it was not possible to do these analyses with a satisfactory number of traces and attacks. What could be done to improve and further confirm the results obtained would be to perform a greater number of attacks on a greater number of traces on ASCON to go and see in more detail the statistical behavior that these implementations have. A further addition that could be made would be to investigate new cryptographic properties to describe more accurately the behavior of S-boxes in real implementations. It

has been seen that nonlinearity and differential uniformity have excellent results in describing the resistance of a function in this case, but a piece is still missing to better describe the behavior for the S-boxes which do not have their optimal value but which still have good resistance.

Appendix A

S-box state of the art

A.1 AES

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	f2	35	3c	a6	e5	c0	8c	bf	0d	a7	ec	08	e4	8b	79	59
10	85	3a	ca	55	de	07	20	fa	11	3e	ff	19	e8	dc	1e	c9
20	91	a5	47	4b	43	db	c7	12	bd	90	a2	be	9e	93	f8	a1
30	c6	6c	1d	b8	a0	7c	fb	4c	d7	86	80	f9	8a	2d	b6	96
40	95	9b	17	e2	6b	40	89	ed	d0	dd	57	00	e0	a3	bb	f4
50	a4	70	63	f1	7d	48	44	6f	0e	af	3d	03	b5	a9	13	d5
60	a8	94	78	aa	18	9f	97	22	7e	81	26	fc	b9	5e	83	62
70	33	9c	2a	ea	50	cd	ee	05	fe	30	04	d3	0b	c3	87	5c
80	74	1c	7a	71	5f	29	28	1f	61	da	df	46	4e	6d	b1	2e
90	6a	5d	65	5a	d6	88	34	f3	32	f6	73	58	69	c2	66	d8
a0	39	d2	4a	99	06	6e	3f	c4	e7	2c	49	3b	c8	d4	f0	d9
b0	d1	4d	b2	0f	92	b7	ae	84	37	14	21	7f	8f	56	8e	98
c0	1a	68	67	e9	9d	25	09	cb	41	ad	cf	51	72	e3	64	16
d0	cc	ba	c5	b4	5b	01	52	f5	c1	fd	77	02	b0	38	27	8d
e0	e6	36	ac	45	0c	75	7b	82	eb	24	42	bc	31	60	9a	76
f0	2b	1b	4f	2f	23	ef	ce	10	b3	0a	54	f7	e1	ab	15	53

Table A.1: *Freyre 1* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	55	38	84	9b	5c	3d	c0	f6	f2	02	c3	4c	bf	c4	e9	c7
10	34	94	9d	8c	ce	ae	ed	86	50	67	69	4a	90	bb	d2	b4
20	11	af	f0	e8	5a	fd	b9	47	b1	95	d9	40	4b	27	be	54
30	9e	14	07	65	7d	89	3e	63	a9	1d	82	2f	1f	78	2a	7e
40	c2	c5	01	6f	b5	da	60	eb	e7	a4	0f	1c	df	19	74	72
50	62	44	6e	80	73	6c	f9	c8	48	b6	33	2b	68	fc	8e	37
60	10	3b	a6	96	c1	cf	57	ea	8a	6a	e3	08	8d	b2	bd	52
70	7a	88	b0	1b	d7	2c	e6	66	91	9a	06	6b	59	17	83	db
80	d5	22	85	4d	fe	0b	ad	f4	56	32	03	5e	b3	dc	26	7b
90	16	cd	4e	2e	21	ff	ac	79	a1	23	ec	04	c6	e4	7f	28
a0	53	39	cb	a0	d4	7c	fb	3a	0c	5d	58	92	05	3f	d6	5b
b0	25	61	12	bc	a3	e1	29	5f	75	41	1a	98	de	51	4f	93
c0	97	24	e2	49	b7	e0	36	8b	b8	d8	18	f1	c9	e5	31	f5
d0	30	a7	43	0e	a8	f7	6d	8f	cc	99	ee	42	d3	1e	f8	45
e0	2d	a2	dd	20	9c	aa	ef	81	64	77	46	0d	13	76	35	d1
f0	d0	71	00	f3	87	a5	15	9f	ab	0a	70	ca	fa	09	3c	ba

Table A.2: *Freyre 2* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	cc	35	88	65	05	c1	0d	91	99	61	4b	f8	fb	ec	77	fa
10	28	92	02	89	13	b0	3b	39	3a	95	b9	33	b8	94	b5	da
20	c7	bf	7a	6f	87	54	34	f6	66	48	10	aa	29	9d	98	d9
30	76	a8	17	09	3f	27	d5	57	3e	24	6c	3c	49	14	8e	42
40	1b	18	43	6a	5b	93	c8	90	ff	fc	f3	e6	a9	8a	f9	2e
50	4a	ae	59	96	44	c5	04	51	2c	4e	9c	f2	1f	4f	bb	9b
60	23	2d	d3	c6	cf	ca	ad	cb	e0	a6	fe	31	74	b3	45	46
70	b6	7d	ee	1e	eb	6b	ea	7f	1c	01	f0	08	4c	1a	6d	2f
80	38	85	82	06	e4	d4	75	69	7e	8f	b1	9f	b2	5d	ba	41
90	56	63	8c	9a	62	0e	c2	80	36	78	0f	1d	4d	fd	de	e7
a0	a3	73	ac	af	d2	2b	67	f7	0b	d7	8d	a7	12	22	0c	53
b0	f4	8b	5a	cd	79	ab	52	19	20	2a	81	84	e8	dc	16	e3
c0	32	e1	68	37	d0	a4	58	72	6e	71	a5	c3	ed	5f	7b	db
d0	00	11	03	50	40	21	25	30	e9	5e	f1	b7	e5	b4	70	55
e0	df	3d	d6	9e	47	5c	c4	97	86	ce	60	c0	15	26	d1	d8
f0	7c	be	a2	64	bc	ef	dd	f5	bd	0a	a1	e2	a0	c9	83	07

Table A.3: *Freyre 3* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	78	2e	f5	01	42	e8	8c	c8	4f	18	1c	7d	b6	98	5c	07
10	9b	24	df	6d	64	cc	7f	74	03	44	b0	0e	9a	b3	9e	54
20	f9	4e	e4	70	13	00	36	08	e0	6b	b2	f1	51	d5	86	46
30	fa	db	31	6c	a2	5f	fe	67	1f	8f	22	5b	33	39	c0	43
40	bc	be	b1	82	e2	10	68	a4	eb	30	4a	47	83	0a	5a	f7
50	7a	ca	f3	a5	3b	9f	4b	c7	e5	32	21	cd	35	60	ce	3d
60	7b	69	fd	41	aa	d4	f0	92	09	84	11	2b	95	bd	27	28
70	4d	9c	0f	a6	14	ea	45	dd	e1	06	8b	56	2d	23	85	1a
80	90	57	91	61	a8	3a	49	97	0d	29	2c	1b	1e	58	e6	48
90	40	8d	55	a0	f2	6e	de	b5	75	3f	ec	a9	cf	d1	a3	c6
a0	79	34	5d	04	ad	b7	20	ae	bb	96	17	9d	99	e9	65	53
b0	72	77	dc	3c	f4	d2	fb	50	d3	81	38	3e	e3	d6	5e	63
c0	bf	87	d8	7e	52	2a	88	80	19	af	c4	59	6a	25	2f	94
d0	ed	15	8e	71	7c	0b	93	ff	cb	ab	c5	ba	d7	4c	c9	89
e0	76	b8	d0	b4	73	8a	e7	fc	c2	62	1d	37	66	c3	12	f6
f0	b9	0c	c1	d9	05	6f	16	ac	f8	ee	da	02	26	a7	ef	a1

Table A.4: *Hussain 6* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	a1	9f	98	82	6c	ea	5a	fc	f0	c2	28	55	cc	39	51
10	95	ce	d6	58	0f	3e	37	69	74	3d	53	e1	4a	87	76	da
20	f9	86	7e	01	02	e3	2c	48	e5	34	c7	1d	e2	ac	45	ee
30	cd	07	2d	20	bb	0a	35	4c	15	1a	af	6b	92	ab	62	a9
40	c8	23	27	43	6e	03	71	aa	7d	05	a5	70	9b	c6	a3	ec
50	fe	61	5b	7b	a8	60	de	f1	7c	1b	44	d4	fb	8d	81	66
60	df	47	d7	3b	ef	22	d3	2b	6d	7a	04	d5	30	90	e4	9e
70	d9	e8	9c	f2	bc	57	93	1c	7f	72	2a	65	54	88	d1	40
80	1f	fd	64	12	b8	5d	e7	0c	78	33	dc	c0	f4	f5	ca	84
90	3f	96	fa	09	8e	36	c1	91	3c	b9	31	d2	32	41	6f	1e
a0	ed	97	b5	2f	73	8f	a0	f6	46	5e	ba	94	b4	bd	3a	f7
b0	6a	18	d0	ae	9d	89	52	0e	db	9a	80	19	16	4b	29	24
c0	8b	08	eb	a4	8c	f8	25	8a	b6	bf	79	ff	d8	b1	0b	4f
d0	a7	17	49	a2	68	56	a6	b2	21	85	4e	38	83	be	b7	2e
e0	42	4d	b3	dd	77	b0	00	e0	cb	c4	e6	67	13	c9	e9	5c
f0	ad	10	c3	c5	14	cf	06	50	5f	59	75	0d	99	f3	11	26

Table A.5: *Ozkaynak 1* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	9a	d9	e3	6e	55	1d	c7	25	44	15	5b	4e	d0	03	94	28
10	c6	34	36	02	49	07	a8	c9	e5	b8	92	06	ac	1c	2c	43
20	c3	35	6a	0a	cc	83	9d	b9	bb	9c	ce	a1	51	67	d3	21
30	60	9f	48	86	a4	8f	8c	c1	91	e7	ed	0c	dd	bc	c5	74
40	2f	13	81	68	33	ec	38	85	37	dc	57	01	cb	75	d2	18
50	04	ae	af	71	22	d5	ab	ff	1e	2b	82	bf	39	89	4c	ea
60	f7	f4	ad	df	3f	3c	e6	a6	08	be	8b	63	31	c8	17	f5
70	3a	66	e2	53	7a	46	f1	5e	7f	29	c2	e9	61	fb	6b	1a
80	6d	3d	f8	5a	c0	a7	93	52	9e	e1	24	32	54	5c	58	26
90	4a	88	8a	e8	3e	b0	80	bd	7c	76	a9	0e	e4	00	f3	b5
a0	7b	fe	14	ca	4b	95	db	78	a0	09	fd	27	b4	cf	72	8e
b0	b7	5d	65	0f	ee	b1	84	d4	23	fa	ef	f9	b3	11	41	ba
c0	0b	7d	b2	2d	aa	8d	79	7e	77	40	90	b6	70	16	a5	de
d0	64	45	fc	d8	0d	1b	98	eb	50	05	c4	3b	19	97	4f	9b
e0	f0	4d	73	47	1f	69	5f	56	d1	96	62	59	a3	f6	42	12
f0	a2	d6	da	2a	f2	2e	6f	30	d7	e0	87	6c	99	20	10	cd

Table A.6: *Azam 1* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	21	97	41	cf	0c	67	60	7b	be	7e	52	9b	15	01	e5	ba
10	3d	e0	2a	b3	3f	b2	49	99	8a	a8	92	29	2e	09	6d	b8
20	7c	f3	ec	39	13	06	64	5e	45	30	74	d8	36	e4	5a	51
30	2f	0d	58	c5	f7	81	ce	c6	dd	05	4e	50	96	c8	91	37
40	3c	69	d4	12	d2	2b	89	fa	87	a6	34	73	5b	d0	19	c7
50	4d	aa	79	7a	0b	fe	1b	9d	af	22	68	c9	5f	de	85	b0
60	24	03	8d	da	1e	a2	dc	c1	1c	6e	df	a1	4a	b6	e2	71
70	00	70	ea	90	f1	14	9c	3e	31	17	1a	23	94	65	e9	38
80	b5	82	76	95	46	ad	47	2d	32	cc	0a	57	e8	5d	b1	43
90	04	78	08	28	48	7d	5c	72	44	53	e1	f6	9e	8f	35	c4
a0	f9	f2	88	c3	a0	d5	83	6b	42	1d	e6	bc	26	6f	cd	fd
b0	ab	fb	66	eb	1f	7f	d9	11	b7	75	25	d3	a4	61	77	db
c0	a7	86	18	10	ff	02	20	d7	e3	9a	bb	4b	e7	f0	ac	8e
d0	f4	59	0e	62	4c	55	93	4f	40	b4	d6	8b	98	ee	33	b9
e0	16	2c	c2	63	27	a9	cb	bd	6c	56	84	ed	a3	ef	d1	f5
f0	3b	ca	0f	3a	f8	80	ae	8c	c0	bf	6a	a5	9f	54	07	fc

Table A.7: *Azam 2* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	0f	0d	f7	f9	a7	b7	b3	ad	65	cc	69	d2	d6	cd	c7	13
10	a4	26	55	48	62	5a	71	0c	ef	d9	a5	e4	7b	c3	1a	d8
20	cf	1e	b6	db	0e	d7	e8	87	f1	91	11	f4	df	72	1d	46
30	68	51	47	63	bf	80	e3	56	ac	b9	05	4b	c5	b8	6d	f8
40	a2	fa	19	6e	7d	e6	81	23	66	ea	36	ab	c2	10	21	49
50	9b	f6	9a	54	95	86	ee	12	f0	43	c8	fd	3d	1f	aa	b4
60	37	14	e0	bb	0a	93	5c	85	c4	f2	92	1b	22	8c	1c	c0
70	3f	7f	8f	cb	89	02	4a	c1	41	04	7c	33	6b	18	2a	7a
80	67	16	29	e2	eb	fc	74	d4	4d	31	30	c9	94	dd	fb	50
90	e5	73	5d	8b	b5	34	61	77	bd	a6	15	2d	35	64	20	83
a0	70	5e	3b	8e	75	24	99	fe	42	9e	4f	79	08	82	84	3c
b0	f5	e7	7e	98	97	59	00	27	a0	88	25	4e	ec	38	ce	9d
c0	de	ae	52	45	06	53	dc	03	39	6f	d0	2f	8d	57	a8	b0
d0	0b	76	a9	3a	f3	78	96	5b	be	17	b2	2c	07	2b	b1	4c
e0	a1	90	a3	44	58	8a	da	6c	9f	ba	28	ed	af	2e	c6	60
f0	ca	09	3e	32	40	e9	ff	d1	bc	01	6a	e1	5f	d5	9c	d3

Table A.8: *Azam 3* S-box

A.2 ASCON

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	01	00	19	1a	11	1d	15	1b	14	05	04	17	0e	12	02	1c
01	0f	08	06	03	0d	07	18	10	1e	09	1f	0a	16	0c	0b	13

Table A.9: *Lut Bilgin* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	10	0e	0d	02	0b	11	15	1e	07	18	12	1c	1a	01	0c	06
01	1f	19	00	17	14	16	08	1b	04	03	13	05	09	0a	1d	0f

Table A.10: *Lut Shamash* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	18	09	1b	06	03	1f	16	01	14	1e	08	05	0a	15	0f	10
01	04	13	17	0c	1c	00	0d	1a	07	0b	19	12	11	14	02	1d

Table A.11: *Lut Lu 4* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	17	1c	0f	10	02	01	15	1e	19	13	12	0c	0b	08	0d	06
01	18	0e	00	03	05	1d	0a	1b	04	07	1f	09	1a	16	14	11

Table A.12: *Lut Lu 5* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	03	0d	1a	16	11	02	0f	15	00	17	0c	09	14	19	1e	0a
01	1b	0e	04	1d	1c	08	01	12	07	18	10	13	1f	06	0b	05

Table A.13: *Lut Lu 6* S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	16	0f	10	09	1b	03	05	06	01	15	1e	12	1c	08	0a	1d
01	0e	00	0d	1a	18	14	11	1f	13	0c	07	19	0b	17	04	02

Table A.14: *Lut Lu 7* S-box

Bibliography

[1] Jean-Philippe Aumasson. *Serious Cryptography : a practical introduction to Modern encryption*. San Francisco, CA: No Starch Press, 2018 (cit. on pp. 2, 8, 38).

[2] John Talbot and Dominic Welsh. «One way functions». In: *Complexity and Cryptography: An Introduction*. Cambridge University Press, 2006, pp. 125–140 (cit. on p. 3).

[3] Claude Shannon. «Communication Theory of Secrecy Systems». In: vol. 28. Jan. 1948. DOI: 10.1002/j.1538-7305.1949.tb00928.x (cit. on p. 3).

[4] Kaisa Nyberg. «Differentially uniform mappings for cryptography». In: *Advances in Cryptology — EUROCRYPT '93*. Ed. by Tor Helleseth. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 55–64. ISBN: 978-3-540-48285-7 (cit. on p. 3).

[5] P. Wright. *Spycatcher: The Candid Autobiography of a Senior Intelligence Officer*. Stoddard Publishing Co. Limited, 1987, pp. 80–85 (cit. on p. 4).

[6] Paul C. Kocher. «Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems». In: *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9 (cit. on pp. 5, 51).

[7] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. «Differential Power Analysis». In: *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25 (cit. on pp. 5, 53, 58).

[8] Paul Kocher, Ruby Lee, Gary McGraw, and Anand Raghunathan. «Security as a New Dimension in Embedded System Design». In: *Proc of IEEE DAC'2004* (Aug. 2004), pp. 753–761. DOI: 10.1145/996566.996771 (cit. on pp. 5, 51).

[9] Wim Eck. «Electromagnetic radiation from video display units: An eavesdropping risk?» In: *Computers Security* 4 (Dec. 1985), pp. 269–286. DOI: 10.1016/0167-4048(85)90046-X (cit. on p. 6).

[10] Christof Paar and Jan Pelzl. *Understanding Cryptography - A Textbook for Students and Practitioners.* Springer, 2010 (cit. on pp. 9, 11, 25).

[11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. «Biclique cryptanalysis of the full AES». In: vol. 7073. Springer, 2011, pp. 344–371. DOI: 10.1007/978-3-642-25385-0_19 (cit. on p. 25).

[12] Mitsuru Matsui. «The First Experimental Cryptanalysis of the Data Encryption Standard». In: vol. 839. 1994, pp. 1–11 (cit. on p. 26).

[13] Mitsuru Matsui. «Linear cryptanalysis method for DES cipher». In: vol. 765. 1994, pp. 386–397 (cit. on p. 26).

[14] Lars R. Knudsen and Matthew J. B. Robshaw. *The Block Cipher Companion.* Springer Publishing Company, Incorporated, 2011 (cit. on pp. 27, 30, 31).

[15] Claude Carlet, Annelie Heuser, and Stjepan Picek. «Trade-Offs for S-Boxes: Cryptographic Properties and Side-Channel Resilience». In: June 2017, pp. 393–414. ISBN: 978-3-319-61203-4. DOI: 10.1007/978-3-319-61204-1_20 (cit. on pp. 28, 30).

[16] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Jan. 2002. DOI: 10.1007/978-3-662-04722-4 (cit. on pp. 28, 30, 31, 33).

[17] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard; Springer-Verlag.* Jan. 1993. DOI: 10.1007/978-1-4613-9314-6 (cit. on p. 29).

[18] Eli Biham. «New Types of Cryptanalytic Attacks Using Related Keys». In: *J. Cryptology* 7 (1994), pp. 229–246. DOI: 10.1007/BF00203965 (cit. on p. 31).

[19] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. «Improved Cryptanalysis of Rijndael». In: *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings.* Vol. 1978. Lecture Notes in Computer Science. Springer, 2000, pp. 213–230. DOI: 10.1007/3-540-44706-7_15 (cit. on p. 31).

[20] Alex Biryukov and Dmitry Khovratovich. «Related-key cryptanalysis of the full AES-192 and AES-256». In: vol. 2009. Dec. 2009, p. 317. DOI: 10.1007/978-3-642-10366-7_1 (cit. on p. 32).

[21] The CAESAR committee. *CAESAR: competition for authenticated encryption: security, applicability, and robustness.* https://competitions.cr.yp.to/index.html (cit. on p. 38).

[22] NIST Lightweight Cryptography Team. *Lightweight cryptography standardization process: NIST selects ascon*. https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon (cit. on p. 38).

[23] Christoph Dobraunig et al. *ASCON v1.2. Submission to NIST*. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf (cit. on pp. 39, 46, 47).

[24] Avik Chakraborti et al. *Structural Classifcation of Authenticated Encryption Schemes*. https://csrc.nist.gov/CSRC/media/Events/lightweight-cryptography-workshop-2020/documents/papers/structural-classification-lwc2020.pdf (cit. on p. 41).

[25] Meicheng Liu and Siang Meng Sim. «Branch Number of the Diffusion Layer». In: *Fast Software Encryption: 23rd International Conference*. 2016, pp. 101–121. ISBN: 978-3-662-52993-5 (cit. on p. 46).

[26] François-Xavier Standaert. «Introduction to Side-Channel Attacks». In: Dec. 2010, pp. 27–42. DOI: 10.1007/978-0-387-71829-3_2 (cit. on pp. 48, 49).

[27] Colin O'Flynn Jasper van Woundenberg. *The hardware hacking handbook : breaking embedded security with Hardware Attacks*. San Francisco, CA: No Starch Press, 2022, pp. 245–322 (cit. on pp. 51, 53, 61).

[28] Eric Brier, Christophe Clavier, and Francis Olivier. «Correlation Power Analysis with a Leakage Model». In: *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 16–29. DOI: 10.1007/978-3-540-28632-5_2 (cit. on pp. 57, 59, 61, 63).

[29] Yunsi Fei, Qiasi Luo, and Aidong Adam Ding. «A Statistical Model for DPA with Novel Algorithmic Confusion Analysis». In: vol. 7428. Sept. 2012, pp. 233–250. ISBN: 978-3-642-33026-1. DOI: 10.1007/978-3-642-33027-8_14 (cit. on p. 67).

[30] Stjepan Picek, Kostas Papagiannopoulos, Barış Ege, Lejla Batina, and Domagoj Jakobovic. «Confused by Confusion: Systematic Evaluation of DPA Resistance of Various S-boxes». In: Dec. 2014, pp. 374–390. ISBN: 978-3-319-13038-5. DOI: 10.1007/978-3-319-13039-2_22 (cit. on p. 67).

[31] Sylvain Guilley, Annelie Heuser, and Olivier Rioul. «A Key to Success: Success Exponents for Side-Channel Distinguishers». In: Dec. 2015. DOI: 10.1007/978-3-319-26617-6 (cit. on p. 67).

[32] Emmanuel Prouff. «DPA attacks and S-boxes». In: vol. 3557. July 2005, pp. 1–8. ISBN: 978-3-540-26541-2. DOI: 10.1007/11502760_29 (cit. on p. 67).

[33] Colin O'Flynn and Zhizhang Chen. «ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research». In: vol. 8622. Apr. 2014. ISBN: 978-3-319-10174-3. DOI: 10.1007/978-3-319-10175-0_17 (cit. on p. 69).

[34] NewAE Hardware Product Documentation. *CW305 Artix FPGA Target*. https://rtfm.newae.com (cit. on p. 69).

[35] Alejandro Freyre Echevarría, Ismel Martínez Díaz, C.M. Legón, Guillermo Sosa Gómez, and Omar Rojas. «Evolving Nonlinear S-Boxes With Improved Theoretical Resilience to Power Attacks». In: *IEEE Access* 8 (Nov. 2020). DOI: 10.1109/ACCESS.2020.3035163 (cit. on p. 70).

[36] Naveed Azam, Umar Hayat, and Ullah. *Efficient Construction of S-boxes Based on a Mordell Elliptic Curve Over a Finite Field*. Sept. 2018. DOI: 10.48550/arXiv.1809.11057 (cit. on p. 71).

[37] Iqtadar Hussain, Tariq Shah, Muhammad Gondal, and Hasan Mahmood. «An efficient approach for the construction of LFT S-boxes using chaotic logistic map». In: *Nonlinear Dynamics* 71 (Jan. 2012). DOI: 10.1007/s11071-012-0646-1 (cit. on p. 71).

[38] Fatih Özkaynak. «Construction of robust substitution boxes based on chaotic systems». In: *Neural Computing and Applications* 31 (Aug. 2019). DOI: 10.1007/s00521-017-3287-y (cit. on p. 71).

[39] *Test Vector Leakage Assessment (TVLA) Derived Test Requirements (DTR) with AES*. https://www.rambus.com/wp-content/uploads/2015/08/TVLA-DTR-with-AES.pdf (cit. on p. 72).

[40] Mathieu des Noes. «Distribution of Signal to Noise Ratio and Application to Leakage Detection». In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024.2 (Mar. 2024), pp. 384–402. DOI: 10.46586/tches.v2024.i2.384-402 (cit. on p. 74).

[41] Kostas Papagiannopoulos, Ognjen Glamočanin, Melissa Azouaoui, Dorian Ros, Francesco Regazzoni, and Mirjana Stojilović. «The Side-channel Metrics Cheat Sheet». In: *ACM Comput. Surv.* 55.10 (Feb. 2023). DOI: 10.1145/3565571 (cit. on p. 76).

[42] Begül Bilgin, Andrey Bogdanov, Miroslav Knežević, Florian Mendel, and Qingju Wang. «Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware». In: Aug. 2013, pp. 142–158. ISBN: 978-3-642-40348-4. DOI: 10.1007/978-3-642-40349-1_9 (cit. on p. 78).

[43] Daniel Penazzi and Miguel Montes. *Shamash (and Shamashash) (version 1)*. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/ShamashAndShamashash-spec.pdf (cit. on p. 78).

[44] Zhenyu Lu, Sihem Mesnager, Tingting Cui, Yanhong Fan, and Meiqin Wang. «An STP-based model toward designing S-boxes with good cryptographic properties». In: *Designs, Codes and Cryptography* 90 (May 2022). DOI: 10.1007/s10623-022-01034-2 (cit. on p. 78).

[45] Petr Socha, Vojtech Miskovsky, Hana Kubatova, and Martin Novotný. «Correlation Power Analysis Distinguisher Based on the Correlation Trace Derivative». In: Aug. 2018, pp. 565–568. DOI: 10.1109/DSD.2018.00098 (cit. on p. 83).