UNIVERISTY OF PIRAEUS - DEPARTMENT OF INFORMATICS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSc «Distributed Systems, Security and Emerging Information Technologies»

ΠΜΣ «Κατανεμημένα Συστήματα, Ασφάλεια και Αναδυόμενες Τεχνολογίες Πληροφορίας»

MSc Thesis
Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**Side Channel Attacks and Countermeasures – Analysis of Secure Implementations** Επιθέσεις Πλευρικού Καναλιού και Αντίμετρα – Ανάλυση Ασφαλών Υλοποιήσεων
Student's name-surname: Ονοματεπώνυμο φοιτητή:	**PASCHALIS KYRANOYDIS** ΠΑΣΧΑΛΗΣ ΚΥΡΑΝΟΥΔΗΣ
Father's name: Πατρώνυμο:	**IOANNIS** ΙΩΑΝΝΗΣ
Student's ID No: Αριθμός Μητρώου:	**ΜΠΚΣΑ/19012**
Supervisor: Επιβλέπων:	**Michael Psarakis, Associate Professor** Μιχαήλ Ψαράκης, Αναπληρωτής Καθηγητής

Ιούλιος 2021 / July 2021

3-Member Examination Committee
Τριμελής Εξεταστική Επιτροπή

Michael Psarakis
Associate Professor
Μιχαήλ Ψαράκης
Αναπληρωτής Καθηγητής

Panagiotis Kotzanikolaou
Associate Professor
Παναγιώτης Κοτζανικολάου
Αναπληρωτής Καθηγητής

Constantinos Patsakis
Associate Professor
Κωνσταντίνος Πατσάκης
Αναπληρωτής Καθηγητής

Preface

One of the courses I was attending in the 1st semester of my postgraduate diploma was "Internet of Things and Embedded Systems". Having already set up my own project of "Smart House", using microcontrollers and sensors, I was keen into the security aspect of it. Towards this direction, I learned a lot in the University's lab. Subsequently, I chose this Thesis to do my own experiments and analysis of secure implementations.

From this position I would like to express my sincere gratitude and appreciation to:

- My supervisor, Dr. Athanasios Papadimitriou, for the most valuable help and guidance during every step of my effort. I also thank him for his time and patience during our long-lasting Skype calls, in the era of COVID-19 pandemic.

- My supervisor, Associate Professor Michael Psarakis, who contributed supplementally to the evaluation of my MSc Thesis.

July 2021

Paschalis I. Kyranoudis

Περίληψη

Τα τελευταία 10 χρόνια ένα συνεχώς αυξανόμενο ενδιαφέρον εκδηλώνεται στο χώρο του Διαδικτύου των Πραγμάτων (IoT). Έχοντας ξεκινήσει σχεδόν 20 χρόνια πριν, οι Smart Cards βρισκόταν σχεδόν σε κάθε πτυχή της ζωής μας (τηλεφωνικές κάρτες προπληρωμένου χρόνου, πιστωτικές κάρτες, συνδρομητική τηλεόραση, κλπ).

Σήμερα, το οικοσύστημα των ενσωματωμένων συστημάτων έχει επεκταθεί δραματικά σε πληθώρα εφαρμογών της καθημερινότητάς μας. Υγεία, μεταφορές, παραγωγή ενέργειας και στρατιωτικής φύσεως υλοποιήσεις, είναι μόνο μερικοί από τους τομείς που εξαρτώνται από την αξιοπιστία και βασίζονται στην ασφάλεια αυτών των συσκευών. Εφόσον αποθηκεύουν και επεξεργάζονται ευαίσθητες πληροφορίες πρέπει να είναι προστατευμένες από μη εξουσιοδοτημένη πρόσβαση. Έτσι, ποικίλα μέτρα ασφαλείας και κρυπτογραφικοί αλγόριθμοι έχουν επιστρατευτεί για να αυξήσουν την ασφάλεια των συστημάτων αυτών.

Ωστόσο, έχουν βρεθεί να είναι εκτεθειμένες σε ένα συγκεκριμένο τύπο επιθέσεων, τις επιθέσεις πλευρικού καναλιού, γνωστές ως Side Channel Attacks. Η επιτυχία και η αποτελεσματικότητά τους βασίζεται στην εκμετάλλευση των ατελειών των ηλεκτρονικών κυκλωμάτων και την εξάρτηση μεταξύ κατανάλωσης ρεύματος και των δεδομένων που επεξεργάζονται οι κρυπτοσυσκευές. Στην παρούσα διπλωματική εργασία θα εστιάσουμε σε υλοποιήσεις του Advanced Encryption Standard (AES), στα αντίμετρα που μπορούμε να ενσωματώσουμε και τα αδύνατά τους σημεία.

Αρχικά, θα περιγράψουμε γιατί οι κρυπτοσυσκευές είναι ευάλωτες σε επιθέσεις πλευρικού καναλιού. Θα αποδείξουμε τους ισχυρισμούς μας εκτελώντας μία επίθεση 1ης τάξης και θα εξηγήσουμε τα αποτελέσματα. Έπειτα, θα αναλύσουμε το αντίμετρο του masking και θα το εφαρμόσουμε στην υλοποίησή μας για να την προστατεύσουμε.

Εφόσον έχουμε ασφαλίσει τον μικροελεγκτή μας, θα επιχειρήσουμε να ξεπεράσουμε το αντίμετρο με μία επίθεση 2ης Τάξης. Επειδή ο όγκος των απαιτούμενων μαθηματικών υπολογισμών αυξάνεται με εκθετικό ρυθμό, θα εξετάσουμε μεθόδους για να κάνουμε αυτές τις επιθέσεις πιο εφικτές για συμβατικά υπολογιστικά συστήματα.

Τέλος, θα συγκρίνουμε τα αποτελέσματα και θα καταλήξουμε σε συμπεράσματα σχετικά με την πολυπλοκότητα και την αποτελεσματικότητα αυτού του είδους των επιθέσεων.

Abstract

Over the past 10 years, a continuously increasing interest has been shown in the field of Internet of Things (IoT) devices. Having started almost two decades earlier, smart cards could be found almost everywhere in our everyday lives (Payphone Cards, ATM/Credit Cards, Computer Security, Satellite TV, etc).

Today, the embedded devices ecosystem has expanded dramatically to multiple areas of our lives. Health, Transport, Energy, Military are just some of the fields that are now heavily dependent on the reliability and security of these devices. Since they hold sensitive data and, sometimes, crucial information, they have to be protected from unauthorized access. Therefore, various security measures and cryptographic algorithms are applied to enhance their security.

However, they have found to be vulnerable to specific kind of attacks, the Side-Channel Attacks, which take advantage of the physical imperfections of the devices and the data-power dependency. In this thesis, we will focus on the implementations that use the Advanced Encryption Standard (AES), the countermeasures that can be integrated and their weak spots.

First, we will describe what makes the cryptographic devices prone to side-channel attacks. As a proof of concept, we will mount a 1st Order Attack and explain the results. Afterwards, we will focus on the Masking countermeasure to secure our device against these types of attacks.

Having our microcontroller set up and secured, as described above, we will attempt to overcome this protection mechanism by exploiting it and executing a 2nd Order Attack, using power traces. As the volume of the needed computations rises exponentially, even for today's standards, we will examine methods to make these attacks more effective and feasible.

Lastly, we will compare the results and draw conclusions about the complexity and effectiveness of the attacks.

Contents

Notation

$\oplus$	Exclusive OR
d_i	The ith byte of the plaintext
k_i	The ith byte of the key
k_{ct}	Correct key
D	Number of plaintexts
T	Matrix of power traces
K	Number of possible key byte values
V	Matrix containing $f(d_i,k)$ for all possible key values
H	Matrix containing hypothetical power consumptions of matrix V
ck	Position where power consumption depends on v_{ck}
v_{ck}	The targeted intermediate value – The output of $f(d_i,k_{ct})$
ρ	Pearson's Correlation Coefficient
u_m	Value *u* concealed by the mask *m*
S(x)	S-Box lookup result of value x
$S_m(x)$	Masked S-Box lookup result of value x
$\dot{t}$	Preprocessed trace
$\dot{T}$	Matrix of preprocessed traces

Glossary

AES	Advanced Encryption Standard
CPA	Correlation Power Analysis
DPA	Differential Power Analysis
FFT	Fast Fourier Transformation
HD	Hamming Distance
HEX	Hexadecimal
HW	Hamming Weight
HO	Higher Order
EM	Electromagnetic
MMIA	Multivariate Mutual Information Analysis
NIST	National Institute of Standards and Technology
POI	Point of Interest
SCA	Side Channel Attack
SPA	Simple Power Analysis
XOR	Exclusive OR

1 Introduction

Side channel attacks are based on information gained from the implementation of a computer system. This thesis is focused on exploiting the leakage of the cryptographic devices. The term refers to devices which run cryptographic algorithms. Practically, these algorithms take as an input some data (plaintext) and a key. By applying multiple mathematic calculations, the algorithm produces the encrypted plaintext, or as we say, the ciphertext. Our goal is to be able to retrieve the key from the device running the cryptographic algorithm.

Typically, breaking a cryptographic algorithm means finding the secret key based on some available information, for example having access to sets of plaintexts and ciphertexts. While brute-force attacks can be mounted in almost any case, we consider an algorithm secure if these kinds of attacks cannot retrieve the key in a reasonable amount of time, using reasonable computing resources.

Since we are focusing on the Advanced Encryption Standard (AES), it is useful to mention that, up to the time writing this thesis, AES is considered mathematically secure. In other words, there is no literature referring to decrypting the ciphertext without having the encryption key.

Before we start exploring the side channel attacks, we will refer to the literature and mention the most important papers and its contributions.

The authors in [1] introduce us to cryptography and cryptographic devices. They describe how power analysis attacks can be mounted and why they work. By using Differential Power analysis (DPA) they are able to recover the key, by using only a few power traces. Moreover, they explain what masking is and how can be applied to AES. Essentially, masking refers to altering the intermediate values of the encryption algorithm, in such a way that the instantaneous power consumption of the device is not related to the actual values anymore. Lastly, they describe, in theory, how this countermeasure scheme can be attacked.

In [2] A. A. Ding *et al.* propose a statistical model for higher order DPA on masked cryptographic algorithms. Their goal is to reveal how higher order attacks work on masked embedded devices and aim to help the system designers regarding their masking implementations.

According to [3], by applying both additive and multiplicative masking, at the cost of a small timing overhead to AES, significant resistance is achieved on higher order attacks. They also describe an affine masking scheme for AES and present their results.

Paul Bottinelli and Joppe W. Bos in [4] analyze the computational aspect of Correlation Power Analysis (CPA) and they propose multiple time-memory trade off techniques to utilize their processing algorithms. Their most important contribution is the incremental Pearson technique, which can compute the correlation coefficient without the need to keep all the traces inside the memory at the same time.

François Durvaux *et al.* [5] propose how the selection of Points of Interest (POIs) can be conducted more effectively. Especially in masked implementations, from the adversary's point of view, finding the points where the leakage is located can be a very time-consuming procedure. They also present two case studies to validate their claims. Since the number of shares is a parameter in their formula, its scope is universal, regardless the number of masks being used.

As described in [6], State-of-the-art masking extraction techniques target the S-Box computation and do not consider the cases where precomputed S-Boxes are stored in the non-volatile memory. Their attack focuses on this gap with a very high success rate, requiring at the same time much less power traces.

In [7], the authors improve the resistance of the high-order masking. They combine three different methods with notable improvement, both in efficiency and speed.

According to [8], if *d* is the number of shares involved in a masking scheme, then an attack with order higher than *d* can be more successful than a *d*-order attack. Additionally, they show that for d=1 the Hamming Weight model is the less preferable to the attacker.

Jiqiang Lu *et al.* [9], explore the AES' rounds and the importance of protecting them with masking. Taking into consideration the 1st and 2nd order attacks, they define the minimum protection against them. This consists of securing the first 2.5 and the last 3 rounds in order to render these attacks ineffective.

The authors in [10] propose a novel Higher Order (HO) attack that takes the pre-processing out of the equation. Their experiments on 2nd and 3rd order Multivariate Mutual Information Analysis (MMIA) showed that, with less than 1000 traces, 100% success rate was achieved. Hence, the need for countermeasures against HO attacks becomes more imperative.

Weijian Li & Haibo Yi [11] presented an improved 2nd Order attack against AES with precomputed masked S-Box. Their adapted CPA technique using the Hamming Weight model is able to reveal the key after 16,000 traces.

Oscar Reparaz et al. [12] deal with the "combinatorial explosion" problem, which is the exponential growth of calculations needed in a higher-order attack. They present a technique to identify the interesting tuples in a "black-box" multivariate environment.

Regarding the bivariate attacks, in [13] a set of preprocessing tools is introduced, which improves the efficiency of these attacks, even if the two leakage points are far from each other. By keeping the analysis in the frequency domain, they can successfully reveal the key in only 3,000 traces, with each of trace being consisted of 4,000 datapoints.

In this thesis, power analysis attacks (a side channel attack variation) and the masking countermeasure will be in the spotlight. We will explain in detail every piece of the puzzle that needs to fall together in order to mount successful attacks. We will begin with AES, refer to the electronic components of typical microcontrollers and their power consumption characteristics. Additionally, we will show experimentally how they can be exploited by using recorded power traces, with the assistance of an oscilloscope. Afterwards, we will switch, from the adversary's point of view, to the defending one's. By using a publicly available Masked AES implementation, the device will be efficiently protected. Next, we will overcome this protection mechanism by escalating our attack to 2nd Order, and we will compare the results. Lastly, our proposal of key-windowing, for the sake of security evaluation and efficiency, will be presented and explained.

The remainder is organized as follows. Section 2 explains what AES is and how its rounds work. Section 3 introduces us to microcontrollers and summarizes the vulnerabilities of cryptographic implementations. Section 4 summarizes the Power Analysis Attacks and describes the necessary steps in order to mount a 1st order attack, whereas in Section 5 we mount a 1st Order CPA attack. Section 6 analyzes the masking countermeasure and presents an existing software implementation on AES. In Section 7 we explain what differentiates a masked implementation, from the attacker's point of view. In Section 8 we mount a 2nd order attack against the masking scheme we described earlier. Last, Conclusions are drawn in Section 9.

2 The AES Block Cipher Algorithm

Rijndael's AES [14] is one of the industry's standards when it comes to symmetric encryption. It was approved by National Institute of Standards and Technology (NIST) in 2001 and belongs to the block cipher group of cryptographic algorithms. This means that it splits the data into fixed-size blocks in order to process them. Their size depends on the key size that is being used. Therefore, AES comes in three flavors with key sizes of 128, 192 and 256 bits. For the sake of simplicity, in this thesis we will focus only on the 128-bit version.

2.1 Structure

AES encrypts a 128-bit data block using a 128-bit key (block). Both the data and the key are represented as a rectangular matrix of 4x4 elements in hexadecimal (hex). In the case of AES-128, ten rounds of specific operations (round transformations) are applied. It is important to mention that each round uses a different key, which is deriving from the initial, using the key scheduling algorithm. In order to decrypt the ciphertext, these keys and round transformations need to be applied in reversed order.

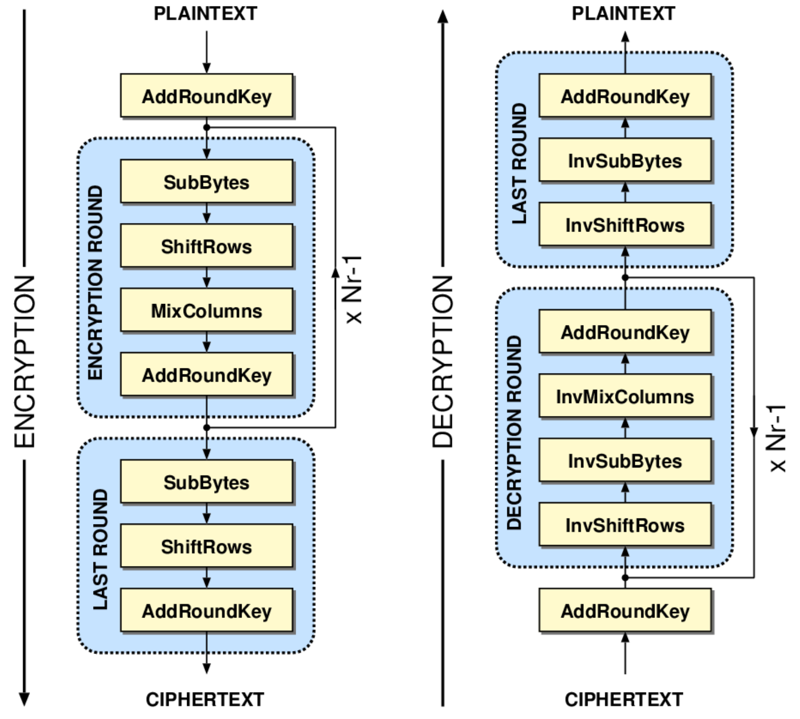


Figure 2.1: AES' Encryption & Decryption Rounds

2.2 Round Transformations

The round transformation is consisted of four different operations, which are applied sequently. These are: AddRoundKey, SubBytes, ShiftRows and MixColumns. In Figure 2.1 we see how the operations and iterations are being performed during encryption and decryption. The “Nr” for AES-128 is 10, meaning that they will be executed 10 times. The only exception is the MixColumns operation, which in the last round is omitted.

2.2.1 AddRoundKey

The key is being added to the state (blocks of plaintexts) using the exclusive-or (XOR) operator (⊕). Since both of them are the same size, the operation is as shown in Figure 2.2

$$\begin{array}{c}\textbf{State}\\ \begin{array}{|c|c|c|c|}\hline 32&88&31&\text{e0}\\ \hline 43&\text{5a}&31&37\\ \hline \text{f6}&30&98&07\\ \hline \text{a8}&\text{8d}&\text{a2}&34\\ \hline \end{array}\end{array} \oplus \begin{array}{c}\textbf{Key}\\ \begin{array}{|c|c|c|c|}\hline \text{2b}&28&\text{ab}&09\\ \hline \text{7e}&\text{ae}&\text{f7}&\text{cf}\\ \hline 15&\text{d2}&15&\text{4f}\\ \hline 16&\text{a6}&88&\text{3c}\\ \hline \end{array}\end{array} = \begin{array}{c}\textbf{Output}\\ \begin{array}{|c|c|c|c|}\hline 19&\text{a0}&\text{9a}&\text{e9}\\ \hline \text{3d}&\text{f4}&\text{c6}&\text{f8}\\ \hline \text{e3}&\text{e2}&\text{8d}&48\\ \hline \text{be}&\text{2b}&\text{2a}&08\\ \hline \end{array}\end{array}$$

Figure 2.2: AddRoundKey Operation

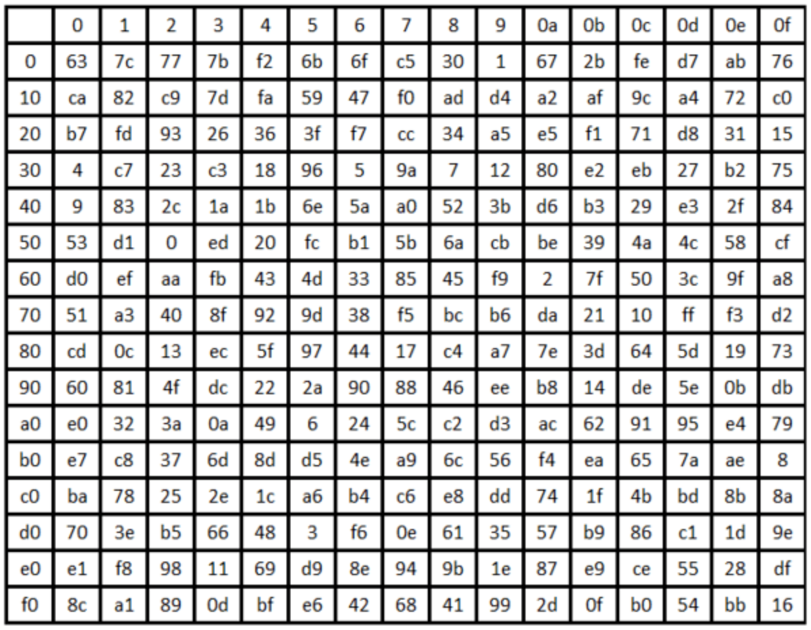
In the above example the output will be a matrix 4x4. The elements of its first line will be: $32 \oplus 2b = 19$, $88 \oplus 28 = a0$, $31 \oplus ab = 9a$, $e0 \oplus 09 = e9$. The rest of the lines will be calculated respectively.

2.2.2 SubBytes

The SubBytes is an operation that substitutes all the elements of the matrix using Rijndael’s S-box [15], which is also known as S-Box or, just, S. Depending on the implementation, it can be either precomputed or calculated “on-the-fly” and it is based on the equation:

$$S(x) = A * x^{-1} + b \qquad (2.1)$$

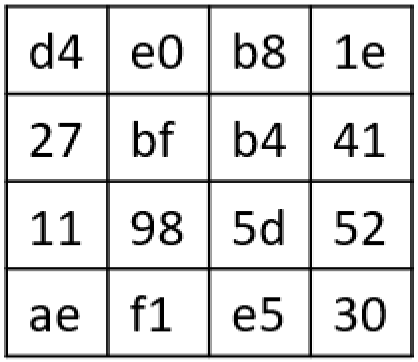
The inverse of *x* is computed over a finite field of 256 elements. *A* declares a matrix and *b* is a vector. This formula has been chosen due to meeting several crucial criteria. Firstly, it is non-linear. This means that the correspondence between the input and output is minimal. Moreover, it is algebraically complex to withstand linear and differential cryptanalysis. Since its computation is resource demanding, it is often precomputed and stored in the non-volatile memory.



	0	1	2	3	4	5	6	7	8	9	0a	0b	0c	0d	0e	0f
0	63	7c	77	7b	f2	6b	6f	c5	30	1	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	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
40	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	2	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	6	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	8
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	3	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

Figure 2.3: Rijndael S-box Table

An example of the S-Box lookup is as follows. In the output matrix of Figure 2.2 we select the hex in the first row and column, which is 19. So, we need the element from the S-Box table in the 1st row and 9th column, which is d4. We will repeat this process for the rest 15 elements. The final matrix is shown in Figure 2.4



d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

Figure 2.4: S-box lookup output

2.2.3 ShiftRows

This operations shifts the positions of the bytes in the matrix. More specifically, the first row remains intact, the second one rotates over 1 byte, the third over 2 bytes and the fourth over 3.

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

Figure 2.5: ShiftRows Output

2.2.4 MixColumns

The mixing of the elements, which applies to the columns of the matrix is called MixColumns. Each column is modulo multiplied in Rijndael's Galois Field by a given matrix, motivated by the so-called wide trail design strategy, which provides high resistance against linear and differential cryptanalysis.

$$\begin{array}{|c|c|c|c|}\hline \mathrm{d4} & \mathrm{e0} & \mathrm{b8} & \mathrm{1e} \\ \hline \mathrm{bf} & \mathrm{b4} & 41 & 27 \\ \hline \mathrm{5d} & 52 & 11 & 98 \\ \hline 30 & \mathrm{ae} & \mathrm{f1} & \mathrm{e5} \\ \hline \end{array} \bullet \begin{array}{|c|c|c|c|}\hline 02 & 03 & 01 & 01 \\ \hline 01 & 02 & 03 & 01 \\ \hline 01 & 01 & 02 & 03 \\ \hline 03 & 01 & 01 & 02 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 04 & \mathrm{e0} & 48 & 28 \\ \hline 66 & \mathrm{cb} & \mathrm{f8} & 06 \\ \hline 81 & 19 & \mathrm{d3} & 26 \\ \hline \mathrm{e5} & \mathrm{9a} & \mathrm{7a} & \mathrm{4c} \\ \hline \end{array}$$

Figure 2.6: MixColumns Operation

2.3 Key Schedule

Key Schedule [16] or Key Expansion is the procedure which generates the keys for each round (round keys). In the first step the key is expanded, and the round keys are extracted from it. The result of the key expansion has a size of 11 x 128 bits. Hence, all the necessary keys are generated for the corresponding 11 AddRoundKey operations. We define:

N as the length of the key in 32-bit words. (Equals to 4 for AES-128)

K_o, K_1, …, K_{N-1} as the 32-bit words of the original key.

R as the number of rounds needed. (Equals to 11 for AES-128)

W_o, W_1, …, W_{4R-1} as the 32-bit words of the expanded key.

RotWord() as a one-byte left circular shift

SubWord() as a function of S-box lookup for every byte of the word.

Then, for *i=0 … 4R-1*:

$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus SubWord\big(RotWord(W_{i-1})\big) \oplus \mathrm{rcon}_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod N \\ W_{i-N} \oplus \mathrm{SubWord}(W_{i-1}) & \text{if } i \geq N, N > 6, \text{and } i \equiv 4 \pmod N \\ W_{i-N} \oplus W_{i-1} & \text{otherwise.} \end{cases}$$

3 Cryptographic Devices

Cryptographic devices are electronic devices which implement cryptographic algorithms and use a secret key, usually stored in non-volatile memory. Although most of us imagine computers to be the most common case that is far from the truth. Smart cards, Microcontrollers and hardware tokens are increasingly and widely adopted.

In order to evaluate the security and reliability of a cryptographic devices we should make assumptions about the knowledge an attacker can have. In this case the safest approach is to apply Kerckhoff's principle. In other words, the attacker knows everything about the device and how the cryptographic algorithm works.

3.1 Components

Cryptographic devices are consisted of various components, each one having its own roles and functionalities. Essentially, we can divide them into two groups. The first one deals with the cryptographic operations, e.g. a digital circuit which performs the encryptions. The second group includes the components which handle the data during the encryption, e.g. non-volatile memory which holds the encryption key. The most typical components of cryptographic devices are:

- **Dedicated Cryptographic Hardware**: A single component includes all the hardware and is exclusively used for performing the cryptographic operations, e.g. an AES encryption chip.
- **General-Purpose Hardware:** This component consists of all general-purpose hardware which are used to perform the encryptions/decryptions, e.g. a microcontroller programmed to implement AES**.**
- **Cryptographic Software:** Includes any type of software that implements a cryptographic algorithm, e.g. Tiny AES [17], which is an AES implementation in C.
- **Memory:** Volatile and not. This component stores the data required by the algorithm, e.g. the cryptographic keys or the intermediate values during an encryption.
- **Interface:** It's purpose is to transfer data from and to a cryptographic device, e.g. a serial connection which feeds the device with plaintexts, in order for them to be encrypted.

3.2 Power Consumption

All digital circuits and their components consume power whenever calculations or operations are performed. Using their power supply to draw current, they transform this electric energy into heat. We distinguish this consumption into two types, Static and Dynamic.

The static power consumption (P_{stat}) includes the power consumptions of the individual components, when the device is idle, meaning that no operation is performed. Generally speaking, the static power consumption is very low.

The dynamic power consumption (P_{dyn}) occurs when a logic cell is flipped (from 0 -> 1 or 1 -> 0). Since the above transitions are data and operation dependent, we can conclude that P_{dyn} is also data/operation dependent. Stated the above, we easily understand that:

$$P_{total} = P_{stat} + P_{dyn} \qquad (3.1)$$

Moreover, since P_{stat} is very low, the dominant factor of P_{total} , is P_{dyn}. Likewise, the total power consumption is data/operation dependent. This constitutes the basic principle of every power analysis attack.

3.3 Power Models & Simulations

In power analysis attacks it is necessary to map the data values being processed with their corresponding power consumption values. However, in order to correlate these values between them, there is no need of knowing the absolute values of the power consumption. Instead, only relative values are being acquired and used. Although, we mentioned earlier that Kerckhoff's principle is applied, the attackers usually have limited information about the attacked device. From their point of view, it is easier to overcome this lack of knowledge by using a power model, compared to studying and analyzing the device's characteristics, which may often require expensive equipment and advanced electronic engineering experience. To our knowledge, the two most widely adopted and used power models are the Hamming Distance and the Hamming Weight.

3.3.1 Hamming Distance

The basic idea behind this model is to count the number of transitions that occur in a digital circuit, given a specific time frame. This way, the number can approach the relative power consumption of the device for this fragment of time. Hence, by splitting the execution time of the algorithm to small time chunks, a kind of power trace can be created. Of course, this power trace will not contain actual power measurements, but just the number or transitions.

It is important to note that when the Hamming Distance (HD) power model is applied, the following assumption are made:

- Both types of transitions (0 -> 1 and 1 -> 0) contribute equally to the power consumption.
- Absence of transitions (0 -> 0 and 1 -> 1) also contribute equally.
- The static power consumption of the cells is not being taken into consideration.

Due to its simplicity, this power model is often used for power simulations. These simulations provide a rough estimation of the power consumption, and it can be calculated relatively quickly.

3.3.2 Hamming Weight

This power model is even simpler than the HD model and is used if the attacker has no information about the device or the preceding and succeeding values of the bus. This power model is accurate when buses of memory elements are pre-charged before writing the new values to them. So, the Hamming Weight (HW) assumes that the power consumption is proportional to the number of bits, in a processed value, which are set to 1. This way, both the data values being processed before and after are completely ignored.

3.3.3 Comparison

Summarizing the above, we can see that HD and HW models are connected. A universally applicable relationship between them is:

$$HD(u_1, u_2) = HW(u_1 \oplus u_2) \tag{3.2}$$

Nonetheless, in order to compare in detail the two power models and draw conclusions about their use cases, we need to take a look into some concrete scenarios. For these scenarios we assume that the attacked device first processes u_1, then u_2 and last u_3 . The goal is to simulate the power consumption of u_2, without knowing the other two values. It is obvious that two transitions take place (u_1 -> u_2 and u_2 -> u_3). We will deal only with the first one, for the sake of simplicity, as the same principles apply for the second transition.

- **Bits of u_1 are equal and constant:** The bus processes a n-bit value with all bits of u_1 being constantly 0. In this case HD equals to HW, as $HD(u_1, u_2) = HW(u_1 \oplus u_2) = HW(u_2)$. In case all bits are set to 1, $HD(u_1, u_2) = HW(u_1 \oplus u_2) = n - HW(u_2)$. Since we are using these models for a power analysis attack, it doesn't matter if the simulated power consumption is proportional or inversely proportional. It is important that, in this case, the HW and HD models are equivalent.

- **Bits of u_1 are constant:** The value u_1 has all its bits constant, but not set to the same value, hence it is not known by the attacker. If we just consider one bit, then the HW and HD are equivalent, just as described in the previous case. If more bits are taken in mind, then the HW model does not describe the u_1 -> u_2 transition very well. However, the more equal bits u_1 has, the better the approach of the HW power estimation is.
- **Bits of u_1 are uniformly distributed and independent of u_2:** This is the worst-case scenario for an attacker trying to apply the HW. Since the bits of u_1 are random and independent of u_2, $HW(u_2)$ is also independent of $HW\ (u_1 \oplus u_2)$. So, HW and HD in this scenario are unrelated.

It is important to mention that although the HD model assumes that both transitions contribute equally to the power consumption, this is not completely accurate. It is shown [1] that a 0 -> 1 transition can lead to a bigger consumption, compared to a 1 -> 0. This means that values with bigger hamming weight lead to bigger power drain. Likewise, HW is, up to a degree, related to the actual power consumption. However, this relationship becomes weaker the more bits of the preceding value are set to 1. Therefore, in a "black box" environment, an attacker should prefer the HD model over the HW, when this is possible.

4 Power Analysis Attacks

Power analysis attacks are a form of side channel attacks. They rely on studying the power consumption of a cryptographic hardware device. As we explained earlier, the physical imperfections of the devices cause a data/power dependency. By observing how the voltage drops or rises during encryptions, an adversary can extract information which, ultimately, lead to the recovery of the key being used by the encryption algorithm.

Typically, the equipment required to mount such attacks includes an oscilloscope, in order to record the power trace(s), optionally an EM probe if we also need to collect EM traces and a computer to communicate with the cryptographic device. Additionally, this computer will run the software which will mount the attack on the recorded power or EM traces.

Based on the individual situation an adversary is (time availability, device characteristics, computing resources, cryptographic algorithm, etc), different approaches of power analysis attacks can be applied.

4.1 Simple Power Analysis

Simple Power Analysis (SPA), as described by Kocher et al. [18], is "a technique that involves directly interpreting power consumption measurements collected during cryptographic operations". In other words, an attacker attempts to extract the key directly from a very small set of given power traces. In the most extreme case, only one power trace is available. We can realize that this can be quite challenging, as in-depth knowledge about the cryptographic algorithm, the device and its instructions are required.

First, a visual inspection of the given power trace(s) should take place. Knowing that, in the AES case, 9 identical rounds (as in the 10th round the MixColumns operation is omitted) are executed (Figure 4.1) [19], we isolate the part of the trace which corresponds to of one of them, e.g. the 1st. This part includes all the 16 bytes of the data and the key being processed.

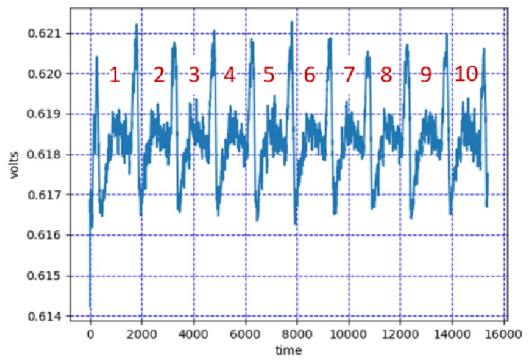


Figure 4.1: Visual Inspection of AES Rounds

Afterwards, we use an identical device we possess and fully control. By feeding this device with multiple plaintexts and keys, we record a large number of power traces. More specifically, we encrypt all possible values of d_i (if it is unknown) with all possible values of the k_i, and record the power traces. Each combination of these values is called template.

Last, we isolate the parts of the power traces that correspond to the same AES round as the initial one and we use statistical methods to calculate their probability. This probability measures how well a template fits with the given trace. Therefore, the highest value corresponds to the better fitting template and, ultimately, to the key byte which was used. Since the algorithm executes the same operations consecutively to all the bytes of the data and the key, by using the same power traces all 16 bytes of the key can be extracted.

4.2 Differential Power Analysis

Differential Power Analysis (DPA) attacks are the most popular type of attacks. This is due to the fact that no detailed knowledge about the device is required. In fact, it is enough to only study the cryptographic algorithm being executed. However, a large amount of power traces is often necessary, especially in a noisy environment, in order for a DPA attack to be successful.

Another important difference between SPA and DPA is that the latter relies on the data dependency of power traces, whereas SPA focuses solely on the time axis in order to find patterns.

A typical DPA attack consists of five steps:

- **Step 1 – Choose an Intermediate Value of the Algorithm:** After studying how the cryptographic algorithm works, we choose an intermediate value v_{ck} (which should be the output of a known function *f(d,k)*, where *d* is a known value (e.g. plaintext) and *k* is the key.

- **Step 2 – Measure the Power Consumption:** We measure the power consumption of the device while it encrypts (or decrypts) ***D*** different plaintexts. For each of these encryptions the attacker should know the plaintext involved. So, we have a vector $d = (d_1,\ldots, d_D)$, where d_i denotes the data of the ith encryption. The corresponding vector of the power traces is $t_i = (t_{i,1},\ldots, t_{i,T})$, where T denotes the length of a trace. We will refer to it as *datapoints*. Since we run D encryptions, the traces synthesize a matrix **T**, sized D x T.

- **Step 3 – Generate Hypothetical Intermediate Values:** We calculate all hypothetical intermediate values for all possible choices of *k*. Knowing that all the operations of AES are executed on the byte level, the possible values for one byte of the key are 2^8 = 256. These values are stored in a vector $k = (k_1,\ldots,k_{256})$, which we will call *key hypotheses.* By applying the known function *f(d,k)* to the 1st bytes of the D matrix, for all *k*, we create a matrix ***V***, with a size of $D \times K$. It should be clear by now that one of the 256 columns of V corresponds to the correct key and the calculation of *f,* that actually took place in the device.

- **Step 4 – Map Hypothetical Intermediate Values to Power Consumption Values:** Using one of the two power models described in the previous chapter, we create a matrix **H**, containing the hypothetical power consumption values, for every element of V. Therefore, the size of H equals to V's.

- **Step 5 – Compare Hypothetical Power Consumptions with Power Traces:** In this final step, each column of H is compared, using statistical analysis methods, with each column of T. This means that we compare the hypothetical power consumptions with the actual consumption, which is imprinted on the power traces. We will store the result in the matrix ***R***, sized K x T. Each element $r_{i,j}$ contains the comparison results between the column h_i and t_j, from matrices H and T, respectively.

At this point, it is important to note that the power traces need to be aligned in order for a DPA to be mounted successfully. Each column t_j of the matrix T must correspond to the same algorithm operation. In a "white-box" or security evaluation environment this can be achieved by setting triggers to the oscilloscope, so in every run the traces start and finish at the same operations. If this is not possible, the traces need to be manually aligned, by inspecting and setting one as a "guide" and adjusting the rest of them to match the guide.

Lastly, we denote as *ct* the position of the power traces where the power consumption depends on the intermediate value v_{ck}. Consequently, the columns h_{ck} and t_{ck} are strongly related, leading to the highest value of the matrix R. We can see the steps 3-5 in Figure 4.2 [1]. Since this value corresponds to an index (from 1-256), the targeted byte of the 16-byte key is retrieved.

4.3 Correlation Power Analysis

The Correlation Power Analysis (CPA) attacks are, actually, very similar to DPA attacks. In fact, the first 4 steps are identical. Even in the last step of a DPA attack we create the same matrices. The only difference lies in the statistical method being used. We utilize the Pearson's Correlation Coefficient (ρ) [20], to determine the linear relationship between the columns as:

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4.1}$$

In this equation n denotes the number of plaintexts (D), $\bar{x}$ and $\bar{y}$ the mean values of h_i and t_j, respectively (i ranges from 1,…,256 and j from 1,…,T). As stated in [21], $-1 \le \rho \le 1$. Value of -1 declares perfectly negative linear correlation, 1 shows perfectly positive linear correlation and 0 stands for no relationship between the two compared variables X and Y.

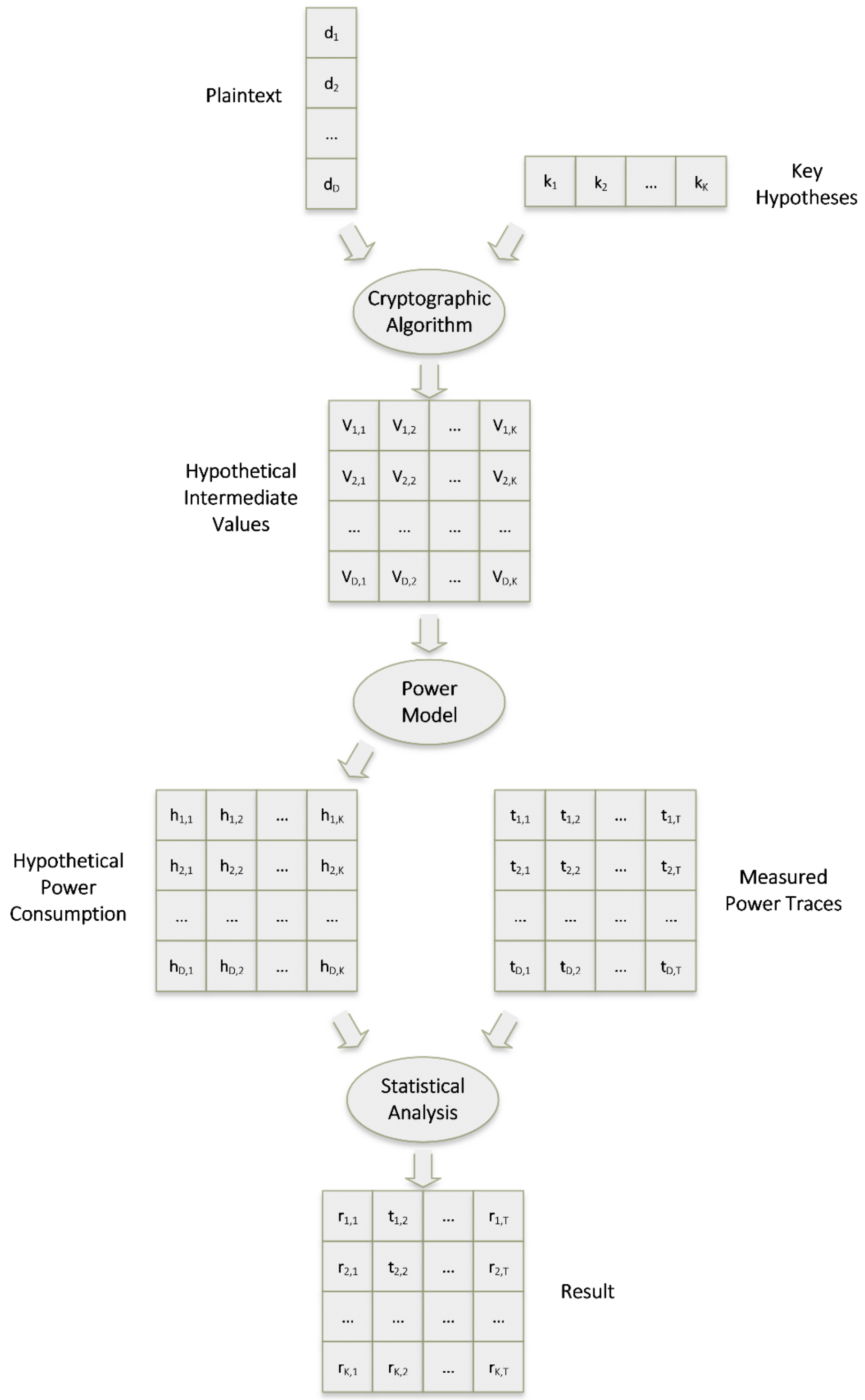

Plaintext
d_1
d_2
...
d_D
Key Hypotheses
k_1 k_2 ... k_K
Cryptographic Algorithm
Hypothetical Intermediate Values
$v_{1,1}$ $v_{1,2}$ ... $v_{1,K}$
$v_{2,1}$ $v_{2,2}$ ... $v_{2,K}$
$v_{D,1}$ $v_{D,2}$ ... $v_{D,K}$
Power Model
Hypothetical Power Consumption
$h_{1,1}$ $h_{1,2}$ ... $h_{1,K}$
$h_{2,1}$ $h_{2,2}$ ... $h_{2,K}$
$h_{D,1}$ $h_{D,2}$ ... $h_{D,K}$
Measured Power Traces
$t_{1,1}$ $t_{1,2}$ ... $t_{1,T}$
$t_{2,1}$ $t_{2,2}$ ... $t_{2,T}$
$t_{D,1}$ $t_{D,2}$ ... $t_{D,T}$
Statistical Analysis
Result
$r_{1,1}$ $t_{1,2}$ ... $r_{1,T}$
$r_{2,1}$ $t_{2,2}$ ... $r_{2,T}$
$r_{K,1}$ $r_{K,2}$ ... $r_{K,T}$

Figure 4.2: Steps 3-5 of a typical DPA attack

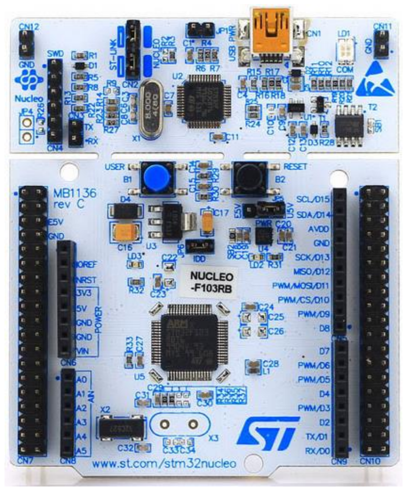
5 1st Order DPA Attack

In the previous chapter we explained how a DPA or CPA attack can be mounted. As a proof of concept, a 1st Order CPA attack on a microcontroller running AES will be executed. Before diving into the technical part of this experiment, we will explain what "Order of an Attack" means.

In section 4.2 we chose an intermediate value to attack (the output of a function *f(d,k))*. Since we are considering only one value, this is referred as a 1st Order Attack. Higher Order attacks will be described and explained in the next chapter. For now, it is enough to say that by increasing the number of intermediate values we target, we also raise the order of the attack.

5.1 Equipment

To mount this attack we will use an STM32 F103RB to perform the encryptions, a digital oscilloscope and a ChipWisperer CW503 probe power supply, in order to record the power traces. The computer which will communicate with the microcontroller and execute the attack hosts an Intel Xeon X5670 with 24GB of DDR3 RAM and is also equipped with an Nvidia GTX1060 6GB GDDR5 GPU.



Processor Type	ARM® 32-bit Cortex®-M3
Clock Frequency	Up to 72 MHz
Internal Memory	128Kb Flash Memory 20Kb of SRAM
Supply Voltage	5 volts
Communication Interface	USB to Serial (ST-Link)

Figure 5.1: STM32F103RB Microcontroller

Table 5.2: Basic Properties of the Attacked Device

5.2 Setup

Given the hardware we will use, its setup is organized as follows. STM32 is connected to a power supply, providing stable voltage. A cable is connected from the trigger pin (as explained in Section 5.2.1) to the input channel No.1 of the oscilloscope. Moreover, a USB connection between the oscilloscope and the computer is made, in order for the traces to be recorded and stored in the computer's hard disk.

In Figure 5.3 we see the schematics of STM32 and we point in red where a known value resistor (e.g. 1ohm) should be connected. We connect one alligator clip from the CW503's probe on each side of the resistor, so the voltage can be measured. One extra cable implements the connection between the CW503 and the input channel No.2 of the oscilloscope. Lastly, one mini-USB cable is connected from the ST-Link interface on STM32 to the computer. This connection will implement the serial bus over which the plaintexts and ciphertexts will be transferred.

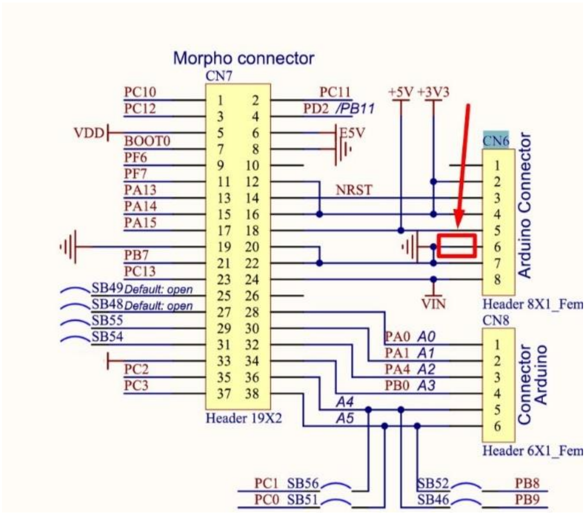


Figure 5.3: Resistor Placement on STM32

5.3 Software

The required software can be divided into two parts. The first one includes the code running in the microcontroller and is written in C language. On the computer side, we will use the MATLAB environment to write the necessary scripts. The reason MATLAB was chosen is because it makes relatively simple to work with matrices and copes perfectly with the microcontroller and the oscilloscope.

5.3.1 Microcontroller

Our microcontroller will run the Tiny AES Project [17], which is an AES implementation, written in C. We will modify it with System Workbench for STM32 [22] and add a few lines of code, which will implement the following:

1. The device will wait for plaintexts. When a plaintext is received via the serial port (from MATLAB) it will be encrypted using a predefined key, stored in the non-volatile memory. The ciphertext will be sent through the serial port (to MATLAB), and the device will wait again, for the next plaintext.
2. We will define a pin as a trigger to the oscilloscope. Afterwards, the trigger is set to start and stop just before and after the S-Box lookup, during the first round of AES. This will be our attack point. Moreover, the traces do not need alignment, as we use the oscilloscope's trigger to synchronize the signals.

The reason behind choosing this exact operation is because it meets the criteria described in the first step of Section 4.2. As we also explained in Section 2.2, the S-Box function is a known operation, which consists of a publicly available table lookup.

5.3.2 Computer

As we mentioned above, we will use MATLAB. The computer plays a double role in this attack. More specifically:

- Our first script will generate a random and fixed size (16 bytes) plaintext and transmit it to STM32. When the ciphertext is received, both the plaintext and ciphertext will be recorded, along with the power trace. Afterwards, the next plaintext will be generated and sent. This procedure will be repeated for 1.000 times.
- The second script will execute the attack. This script follows all the principles and steps mentioned in the previous chapter and was provided by the University's Lab.

5.4 Power Traces

The collected 1.000 power traces are stored in the matrix T, sized 1.000 x 2.000. The 2.000 corresponds to the number of datapoints (or columns) of a single trace. We set the oscilloscope sampling rate low, so we don't get a lot of information to process, as the correlation calculation can be quite resource demanding.

5.5 Mounting the Attack

The attack will be executed to the 1st byte of the key and plaintexts. Using the same power traces and exactly the same procedure, all 16 bytes of the key can be recovered. Since we have chosen already the intermediate value to attack, we will calculate all the possible outputs of S-Box (hypothetical intermediate values) for each plaintext. So, for i = 1,…,1000 and j= 1,…,256 we will generate the matrix V (1000 x 256) as:

$$V_{i,j} = SBox(d_i \oplus k_j) \tag{5.1}$$

We denote *d*d_i as the 1st byte of the ith plaintext and k_j as the 1st byte of the hypothetic key. The reason we are using the XOR operator and the S-Box lookup is due to AES' structure, and more specifically, its AddRoundKey and SubBytes functions.

Thereafter, we will map the matrix V to hypothetical power consumption values. We chose to use the Hamming-Weight (HW) model, for the sake of its simplicity. In order to save execution time, we will store all the matrices into the GPU and use its cores for all the calculations. We have also generated a matrix, containing all the hamming-weight numbers for all 256 possible byte values. This way, the power model calculations will not take place on-the-fly. On the contrary, the mapping will be executed as a simple table lookup, using the precomputed HW matrix. Lastly, we will use MATLAB's built-in function to compute the correlations for all hypothetical key values.

5.6 Results

The execution time was 0,86 seconds for the calculations and another 1,32 seconds for plotting the correlation results. During the attack, RAM usage rose by 400 MB and GPU RAM by 600 MB. As we can see in Figure 5.4a, the higher correlation value is 0,473 and corresponds to index 93. Due to MATLAB's indices starting at 1 instead of 0, we conclude that the correct key index is decimal 92 (which corresponds to the backslash character "\" on the ASCII table) [23].

In Figure 5.4b we see with grey all the hypothetic keys but the real, which is colored in blue. Although we run 1.000 encryptions, we can see that the correlation was over 0,5 after the first 100 traces. Thus, we broke AES with less than 100 traces.

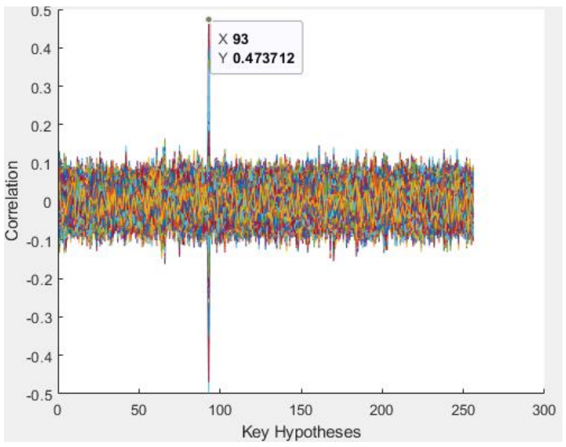


Figure 5.4a: Correlation – Key Hypotheses Plot

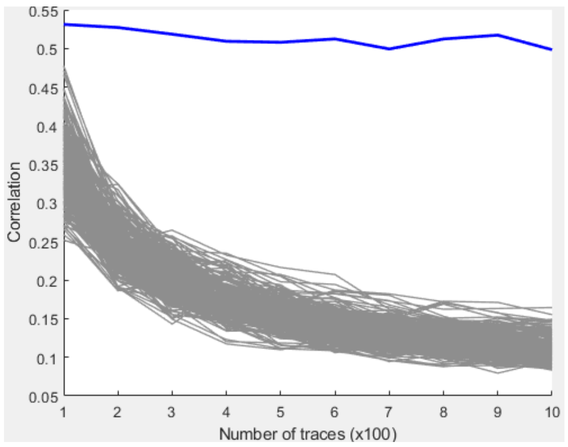


Figure 5.4b: Correlation – Number of Traces Plot

6 Masking

As we showed in the previous chapter, unprotected devices running cryptographic algorithms can be easily exploited and have their keys retrieved. In order to reduce, or even eliminate, these cases various methods have been proposed and applied. The goal of all these countermeasures is to make the power consumption of the cryptographic devices independent of the intermediate values of the algorithm.

Masking accomplishes this by randomizing the values that are processed. A great advantage of this technique is that it can implemented at the software level, without the need of modifying the device's physical characteristics.

6.1 Description

In a masked implementation, each protected intermediate value is concealed by a mask m. This mask has a random value, is generated by the device itself and changes in every execution. Hence, the attacker cannot know or predict the masks. We denote as u_m the value u concealed by the random mask m:

$$u_m = u * m \tag{6.1}$$

The operation $*$ varies according to the chosen masking implementation. As we will explain later in this chapter, it is most often exclusive-or ($\oplus$), modular addition (+) or modular multiplication (x).

It is important that once an intermediate value is masked, it should stay masked all the time. To achieve it, the cryptographic algorithm should be adapted to apply and keep track of all the masks. Once the encryption ends, the masks should be removed in order to obtain the correct ciphertext.

As it is easily understandable, this masking and unmasking procedures adds some overhead, due to the additional computations needed. In order to keep the performance at an acceptable level, the number of masks applied should be chosen carefully.

6.2 Types of Masking

There are two major types of masking: Boolean and Arithmetic. In the first one, the intermediate value is concealed by XOR-ing the value with the mask, on the bit level. In this case we have:

$$u_m = u \oplus m \tag{6.2}$$

On the other hand, in arithmetic masking we use either modular addition or multiplication. The modulo is defined by the cryptographic algorithm. However, there are cases where both types of masking are required. This originates from the nature of the algorithm, which is based on both boolean and arithmetic operations. Studies have shown [24],[25] that this arises problems, because switching between different types of masking usually requires a significant amount of additional calculations.

In this thesis we will focus solely on boolean masking, and more specifically on an AES masking implementation, which will be analyzed later in this chapter.

6.3 Shares

As we explained before, in the case of boolean masking, a masked value $u_m = u \oplus m$. In order to remove the mask and retrieve the initial value, u_m and m must be given. In other words, this intermediate value u is represented by two shares (u_m, m). Knowing only one of them, gives no information about u. Hence, masking is defined as a secret-sharing scheme that uses two shares.

Applying more than one mask on the same intermediate value, and keeping track of all the masks and shares involved, increases the cost of the implementation. Such an approach requires more memory to store the shares and more computing time, in order to compute them. Thus, in practice masking schemes are most often using just two shares.

6.4 Security

The reason why DPA attacks are effective is because at some point, during the execution time of the algorithm, the power consumption of the device depends on an intermediate value which is processed. Masking removes this dependency by altering these intermediate values with the use of masks. Hence, the actual processed value cannot be guessed by the attacker, as the mask is unknown to him.

In order for masking to achieve its goal, the masks should be chosen carefully. The theory is that if u_m is independent of u, then the power consumption of u_m, is also independent of u. We can conclude that if each intermediate value u_m is pairwise independent of u and m, then masking provides resistance against 1st Order DPA attacks.

In typical masking implementations, each masked value induces a distribution which does not depend (statistically) on the unmasked value. Likewise, the distribution of $u \oplus m$, is always the same, regardless the value of u.

It is shown in [26] that, in case of several different masks applied for each protected intermediate value, resistance against Higher Order DPA is achieved. More specifically, n masks can prevent up to a n^{th} Order Attack.

6.5 AES Masking

In this section we will present a software masked AES implementation, as proposed in [1]. This scheme uses exclusively boolean masking and it is tailored to AES structure, as analyzed in Chapter 2. Some of the masks are applied to the state (plaintext) and some others on the first round key. Hence, the whole key schedule operation is masked, as well. Regarding the other operations:

- **AddRoundKey:** This operation XOR-es the state with the key. Since the key is masked, this mask is also applied to the state: $d \oplus (k \oplus m) = (d \oplus k) \oplus m$.
- **SubBytes:** Constitutes the only non-linear operation of AES. As it is a table lookup, we will generate a masked S-Box table.
- **ShiftRows:** This operation moves the bytes of the state. Because at this point the state is already masked, there is no need to apply further masks.
- **MixColumns:** This operation requires more attention, because it mixes the bytes from different rows of a column. Thus, it requires at least 2 masks and we need to make sure, at the same time, that all intermediate values stay masked. On the other hand, it is preferred to mask each row with a different mask. This way, the same masks are used in every round and the output masks are also the same. Hence, we keep the number of different masks used low.

Now, we will put all the pieces of the puzzle together. We will use 10 different masks. The first 2 masks m and m' are the input and output masks of the SubBytes operation. Afterwards we will generate a masked S-Box table, denoted as S_m, such that $S_m(x \oplus m) = S(x) \oplus m'$. The next 4 masks ($m_1$, m_2, m_3, m_4) are the input masks of MixColumns operation. The last 4 masks (m_1', m_2', m_3', m_4') are the output masks of MixColumns and they are deriving from the previous four, by applying a masked S-Box lookup.

A masked AES rounds works as follows. At the beginning of each round the plaintext is masked with m_1', m_2', m_3', m_4'. The round key is masked with $m_1' \oplus m$, $m_2' \oplus m$, $m_3' \oplus m$ and $m_4' \oplus m$. The AddRoundKey operation changes the masks of the state to m. This happens because $m_i' \oplus m_i' \oplus m = m$ (i = 1,2,3,4). Next, the SubBytes operation, using the masked S-Box, changes the masks to m', as $S_m(x \oplus m) = S(x) \oplus m'$. ShiftRows has no effect on the masks, as mentioned before. At this point we will apply the *Remasking*, meaning that the masks of the state will change from m' to m_i for each row, with *i* declaring the number of the row. Lastly, the MixColumns operation will change these masks from m_i to m_i'.

We can observe that, in the end of the round we described, the state's masks are exactly the same as in the beginning of this round. This way, we can apply this masking scheme as many times as required by AES. In the last encryption round, where MixColumns is omitted, the last AddRoundKey operation will remove the masks and we will have the unmasked ciphertext. A schematic of the described procedure and the masks applied in each step can be seen in Figure 6.1.

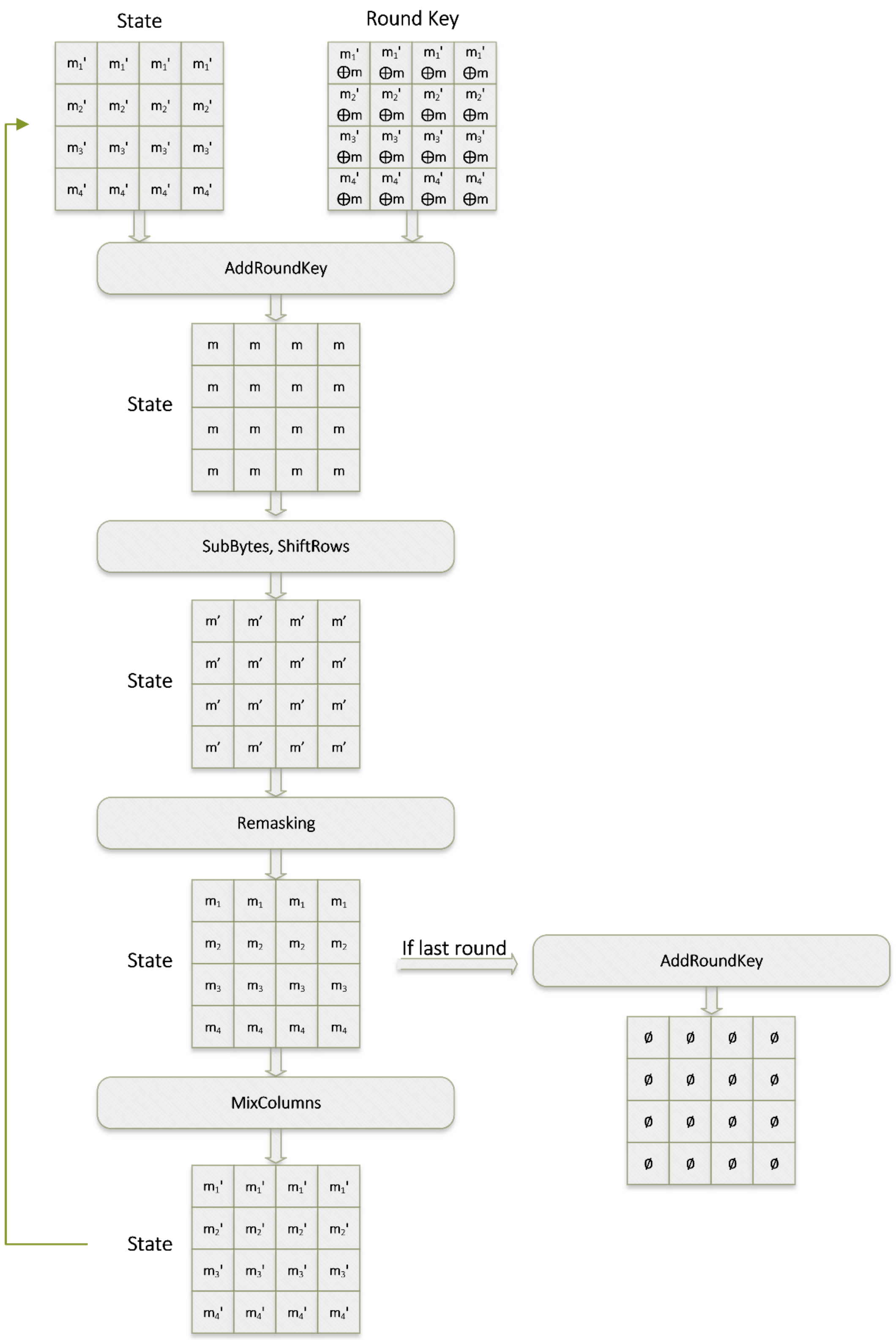


Figure 6.1: AES Masking Scheme

7 Attacks on Masking

Higher order attacks exploit the combined leakage of several intermediate values that are being processed by the cryptographic algorithm. As we mentioned in the previous chapter, typical masking schemes reuse the same masks, with one mask being applied each time. Even today, in implementations where efficiency (speed, memory, low power consumption) is needed, this type of masking is preferred. Although, additional countermeasures against DPA attacks may be applied, when it comes to masking, it is sufficient to concentrate on 2nd Order Attacks, which exploit the joint leakage of two intermediate values. These values can be either two values concealed by the same mask, or a masked value and its corresponding mask.

In the previous chapter we explained what the masking countermeasure is and how it can be applied in AES. In this chapter we will describe the principles and the theory behind 2nd Order Attacks.

7.1 Description

Second-Order DPA attacks exploit the joint leakage of two intermediate values. Generally speaking, this leakage cannot be exploited directly, because it takes place in different operations of the algorithm. Hence, the targeted 2 values are being processed at different times. In this case, it is necessary to preprocess the power traces in order to locate the power consumption values which depend on both intermediate values. Afterwards, we apply a 1st Order attack on these preprocessed traces. More specifically, in the 1st step of a 2nd Order DPA attack we choose two intermediate values, concealed by the same mask, let them be *u* and *v*. Because they are masked, these values occur in the device as u_m and v_m. In the next step, we record the power traces the same way as if it were a 1st order attack. In step 3 we will do the traces' preprocessing. The result of this step is a preprocessed trace, denoted as $\dot{t}$. The matrix which will contain the preprocessed traces for all *D* plaintexts is denoted as $\dot{T}$. Step 4 includes the calculation of hypothetical values, which are a combination of *u* and *v*: *w = comb(u,v)*. Since the attack will be mounted on a boolean masking scheme, this *comb* function will be just the XOR operation:

$$w = u \oplus m = (u_m \oplus m) \oplus (v_m \oplus m) = (u_m \oplus v_m) \oplus m \oplus m = u_m \oplus v_m \quad (7.1)$$

In the above equation we showed how <u>we are able to calculate the hypothetical values of the masked values, without the need to know the masks</u>. In the next step we map *w* to hypothetical power consumption *h.* Last, in step 6 we compare the hypothetical power consumptions with the preprocessed traces. Consequently, we need to find a preprocessing function which maximizes the correlation equation:

$$\rho[HW(u \oplus v), \dot{\mathrm{T}}] \quad (7.2)$$

7.2 Preprocessing Function

In order to select the appropriate preprocessing function, we need to take into consideration that for the correct key hypothesis k_{ck} the above equation is maximized at some point $\dot{t}_{ct}$. This point results from two datapoints in the initial, unprocessed, trace. These instantaneous and exploitable power consumptions occur when each of the two intermediate values are processed.

In an ideal scenario we would compare the hamming weights of the hypothetical consumptions with the hamming weight of the actual computations. Unfortunately, this is not possible because we do not know the actual values being processed. However, applying some operations between hamming weights for hypothetical values gives us indications about the preprocessing's theoretical performance. This is justified, because hamming weights are, up to a degree, related with the actual power consumption (See also Section 3.3.3).

Various types of preprocessing functions have been proposed in the scientific literature. In [26], Chari *et al.* introduce $pre(t_x,t_y) = t_x \, . \, t_y$, whereas in [27] the author presents the absolute value of the difference of the two points: $pre(t_x,t_y) = | \, t_x - t_y \, |$. In Table 7.1 we compare the reliability of several proposed preprocessing functions for a Single-bit scenario. For both values of a bit (0,1) we calculate the HW(u ⊕ v). The next 5 lines include the preprocessed function being evaluated, based on the value of ρ. We can see that the absolute difference performs the best, with ρ=1.

	Value				Correlation
u_m	0	0	1	1	
v_m	0	1	0	1	
HW(u ⊕ v)	0	1	1	0	
$HW(u_m) . HW(v_m)$	0	0	0	1	ρ = -0.57
$\|HW(u_m) - HW(v_m)\|$	0	1	1	0	ρ = 1
$(HW(u_m) + HW(v_m))^2$	0	1	1	4	ρ = -0.33
$HW(u_m) + HW(v_m)$	0	1	1	2	ρ = 0
$HW(u_m) - HW(v_m)$	0	-1	1	0	ρ = 0

Table 7.1: Single Bit Correlation Performance

Likewise, a Multiple-Bit scenario is evaluated in Table 7.2. The performance drops as more bits are added in the equation. Since AES operates on the byte level (8 bits), the maximum theoretical value for the correlation coefficient is 0.24, and is given, once again, by the absolute difference function.

	Number of bits of u_m and v_m			
	1	**2**	**4**	**8**
$HW(u_m) . HW(v_m)$	-0.58	0.32	-0.17	0.09
$\|HW(u_m) - HW(v_m)\|$	1.00	0.53	0.34	0.24
$(HW(u_m) + HW(v_m))^2$	-0.33	-0.16	0.08	-0.04
$HW(u_m) + HW(v_m)$	0.00	0.00	0.00	0.00
$HW(u_m) - HW(v_m)$	0.00	0.00	0.00	0.00

Table 7.2: Multiple Bit Correlation Performance

8 2nd Order Attack

In this chapter we will attempt to mount a 2nd Order Attack against a masked AES implementation. This code will run on the same hardware (STM32 microcontroller) we described in Section 5.1. Moreover, the setup will be exactly the same, as the way of executing the attack is, up to the step of collecting the power traces, identical with the 1st Order Attack.

8.1 Software

The software the microcontroller will run (Masked AES) is implemented by CENSUS, and is publicly available at [28]. It uses the same principles and approaches described in this thesis, and also in [1]. The two intermediate values we will target are the S-Box output of two consecutive bytes during the first round of AES. They meet the criteria described in Section 7, as they are both values concealed by the same mask.

After inspecting the code and its iterations, we noticed that the state's and the key's matrices are processed line by line. This means that after the 1st byte, the next being processed is the 5th, instead of the 2nd. Hence, we target the 1st and 5th bytes of the data and the key. Additionally, in order to reduce the number of the traces, we will set the trigger of the oscilloscope to start and stop exactly before and after the S-Box lookups. In Figure 8.1 we see a plot of the power consumption for 100 plaintexts. On *x*-axis we see the number of datapoints (14.000) and on y-axis the relative power consumption. Intuitively, we conclude that the two spikes are when the lookups take place.

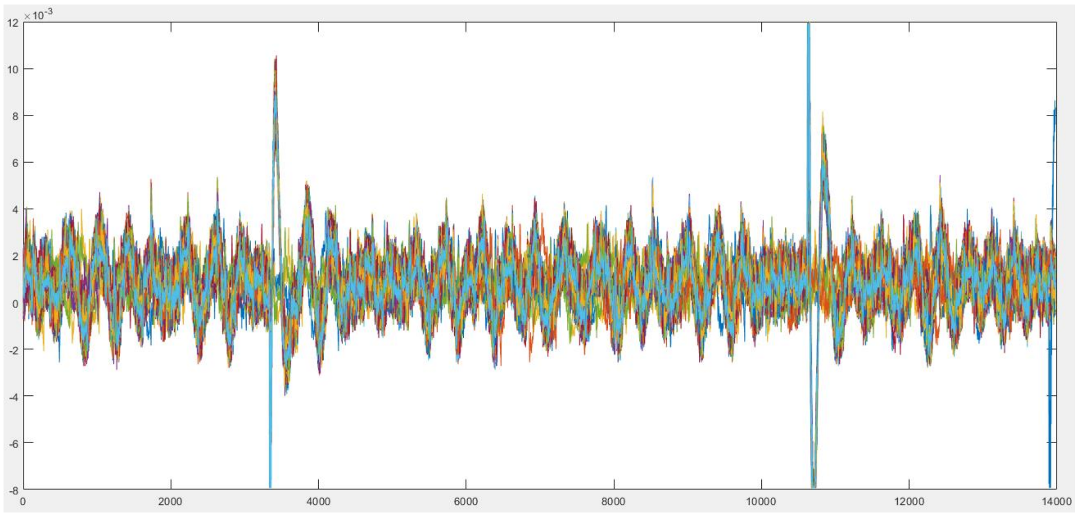


Figure 8.1: Power Consumption Plot for 100 plaintexts

In order to preprocess the traces, with the absolute difference method, we need to take all possible combinations of these datapoints. For 14.000 datapoints the possible combinations are 14.000 * (13.999 / 2) = 97.993.000, corresponding to approximately 70GB of RAM, for 100 traces. Hence, we need to compress the traces in order to reduce the number of datapoints. In Figure 8.2 we compressed the traces and added the synchronized trigger signal (on the top). This will give us indications about where the processing of the intermediate values took place.

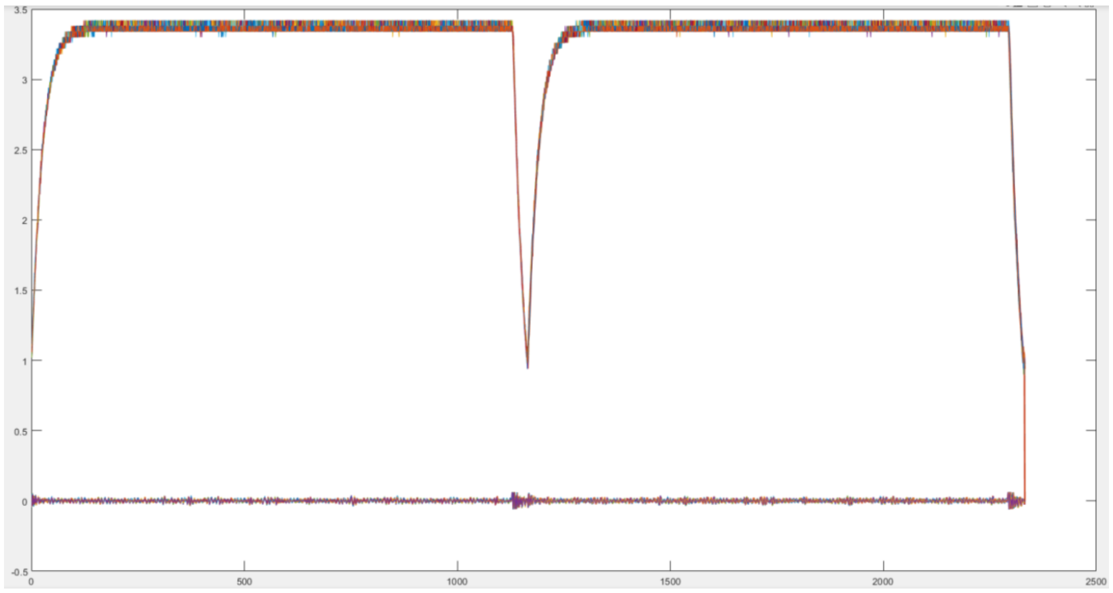


Figure 8.2: Trigger and Consumption Signals

8.2 Incremental Processing

Even after compressing the number of datapoints by a factor of 6, the possible combinations, for all the plaintexts needed by the attack, are forbidden for any conventional computer system. We take into consideration that any further compressing may lead to the absence of leakage points, as the compression (essentially, under-sampling) is lossy. Thus, we will implement an incremental way of a 1st Order Attack, as introduced in [4].

In detail, we need to decompose the built-in Pearson Correlation Coefficient function in MATLAB and calculate the products and mean values in separate variables. It is known that the Eq. (4.1) can be rewritten as:

$$\rho(X,Y) = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n\sum_{i=1}^{n} {x_i}^2 - \left(\sum_{i=1}^{n} x_i\right)^2}\sqrt{n\sum_{i=1}^{n} {y_i}^2 - \left(\sum_{i=1}^{n} y_i\right)^2}} \quad (8.1)$$

Consequently, in order to calculate ρ, we need to compute the following 5 values:

$$s_1 = \sum_{i=1}^{n} x_i\,, \qquad s_2 = \sum_{i=1}^{n} x_i^2\,, \qquad s_3 = \sum_{i=1}^{n} y_i\,, \qquad s_4 = \sum_{i=1}^{n} y_i^2\,, \qquad s_5 = \sum_{i=1}^{n} x_i y_i$$

Once they are computed, Eq. (8.1) can be rewritten as:

$$\rho(s_1, s_2, s_3, s_4, s_5) = \frac{(ns_5 - s_1 s_3)}{\sqrt{(ns_2 - s_1^2)(ns_4 - s_3^2)}} \tag{8.2}$$

The above products can be calculated in an incremental way. We will load batches of power traces into RAM, computed these products, free the memory and load the next batch. The only things we need to keep are these 5 variables. Essentially, there is no limit in the number of the power traces we can use to mount the attack.

8.3 Security Evaluation

Before attempting a 2nd Order attack, it is crucial to determine if the masking implementation we chose actually resists against a typical 1st Order CPA. Using the incremental MATLAB script, we target just one intermediate value and we will perform the attack. After 5.000.000 traces, the correct key remains hidden in the correlation charts. Please note that all the attacks we conducted so far take place in the time domain, where each datapoint represents a fragment of time.

The authors in [29] propose applying Fast Fourier Transformations (FFT) in CPA, in order to improve the effectiveness of an attack. Thus, we applied this frequency-domain approach in our attack script. The results can be seen in Figure 8.3a and 8.3b. After almost 3.000.000 traces the correct key stands out in the correlation plot. However, 3 million traces is an extremely high number, hence we consider that the Masked AES resists a 1st Order CPA attack successfully.

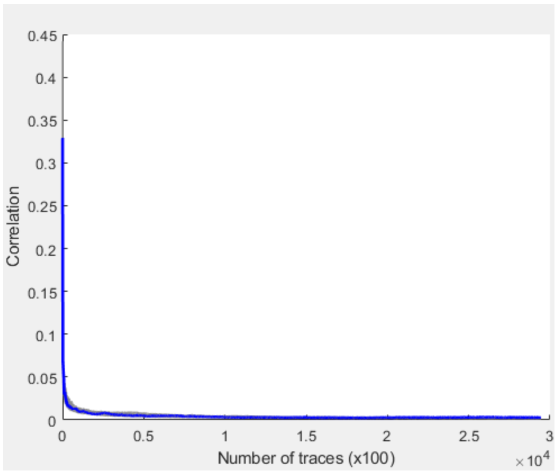


Figure 8.3a: FFT Correlation Plot

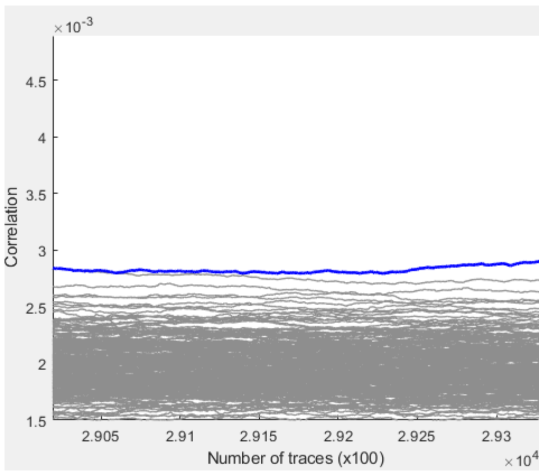


Figure 8.3b: FFT Correlation Plot – Zoomed View

8.4 Mounting the Attack

In a 1st Order attack the possible values of the key are 256. As we explained in the first section of this chapter, when it comes to a 2nd Order Attack, two intermediate values (plaintext bytes) and two key bytes are targeted. This increases the number of possible key values to 256^2 = 65.536. These are the combination of keys we need to examine in order to find the correct keys. Please note that our attack will reveal 2 bytes of the key, as the highest correlation index will point to a matrix consisted of key combinations. The highest scoring index will correspond to two bytes of the key.

In the computer system we are using, even with the compressed traces, the usage of RAM stays out of bounds. Hence, we need to use some kind of windowing, as it has been proposed in the scientific literature. This consists of splitting each unprocessed power trace into small windows of, e.g. 50 datapoints. For a total of 2300 datapoints we will get 46 windows. All the possible combinations of these windows are 46 * (45 / 2) = 1035. However, as the recorded and targeted operation of AES consists of two identical S-Box lookups, we can safely assume that the power trace is symmetric to the middle of the trace. Hence, there is no benefit into combining windows which belong to the same half of the trace. This will reduce the number of combinations to $1035 - \left(\frac{46}{2}\right)^2 = 506$.

From each valid combination of the traces a preprocessed trace will be generated and the attack will be mounted on it. The window of 50 traces was chosen, as the resulting batch of 100 preprocessed traces fits into the 6GB of GPU RAM, and we will be able to mount the attack more effectively. On the other hand, instead of this approach, we could apply windowing on the key hypotheses matrix. This is a simpler implementation, as we split the matrix of 65.536 keys into chunks of, so to say, 100 keys and mount attacks, for a fixed number of traces (e.g. 10.000). Once each batch of keys is processed, we will move the window to the next hundred of keys. However, we might mistakenly assume that 10.000 of traces are enough for the CPA to work. In case none of these windows reveals the key, we need to start from the beginning and increase the number of traces.

Nonetheless, we are in a “white-box” environment, and our goal is to evaluate if our masked AES implementation can be broken. Therefore, our key hypotheses matrix will contain the correct combination of the 1st and 5th key bytes, and another 2 false combinations. If our attack works, the points where the leakage is will be revealed.

As expected, the correct key combination stands out among the other two and we get an indication about the datapoints where the leakage is. In Figure 8.4 we mark the two points where the 2 leakages of the targeted intermediate values are.

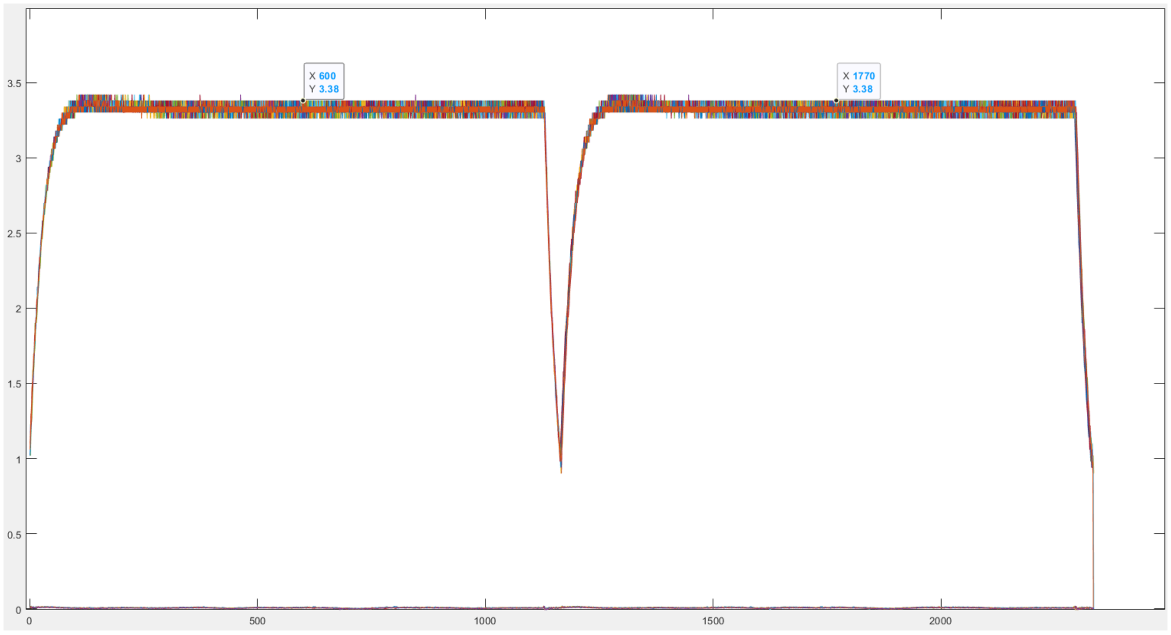


Figure 8.4: The 2 Leakage Points

Now that we got an “educated guess” about the location of the leakage we can execute the attack again using all possible 65.536 keys and target only the windows which contain the leakage points.

8.5 Results

In our experiments we targeted 10, 5 and 2 datapoints from the preprocessed traces. The execution time of the attack was 3’ 20”, 1’ 50” and 58”, respectively, for a total of 10.000 traces. In all 3 cases the key index was recovered successfully, even when using just 2 datapoints from the initial trace of 2300! In Table 8.5 we can see more detailed performance results.

Number of Datapoints	Execution Time	RAM	GPU RAM	Max. Correlation
2	58”	100MB	150MB	0.0877291
4	1’ and 50”	150MB	200MB	0.0877291
10	3’ and 20”	200MB	250MB	0.0877291

Table 8.5: 2nd Order Attack Performance

In Figures 8.6 the correct key (in blue) stands out after almost 5.500 traces, whereas in Figure 8.7 we see the index of the recovered key (64558). This index corresponds to the 1st key byte decimal 252 (character “ü“) and 5th key byte decimal 45 (character “-“)[23].

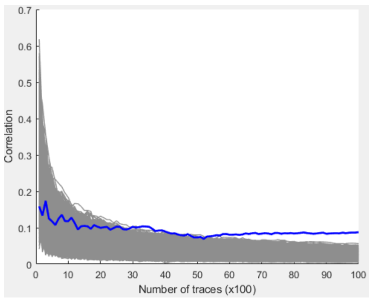


Figure 8.6a: 10 Datapoints

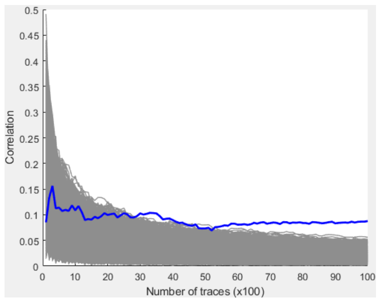


Figure 8.6b: 4 Datapoints

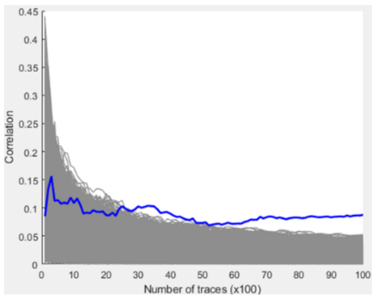


Figure 8.6c: 2 Datapoints

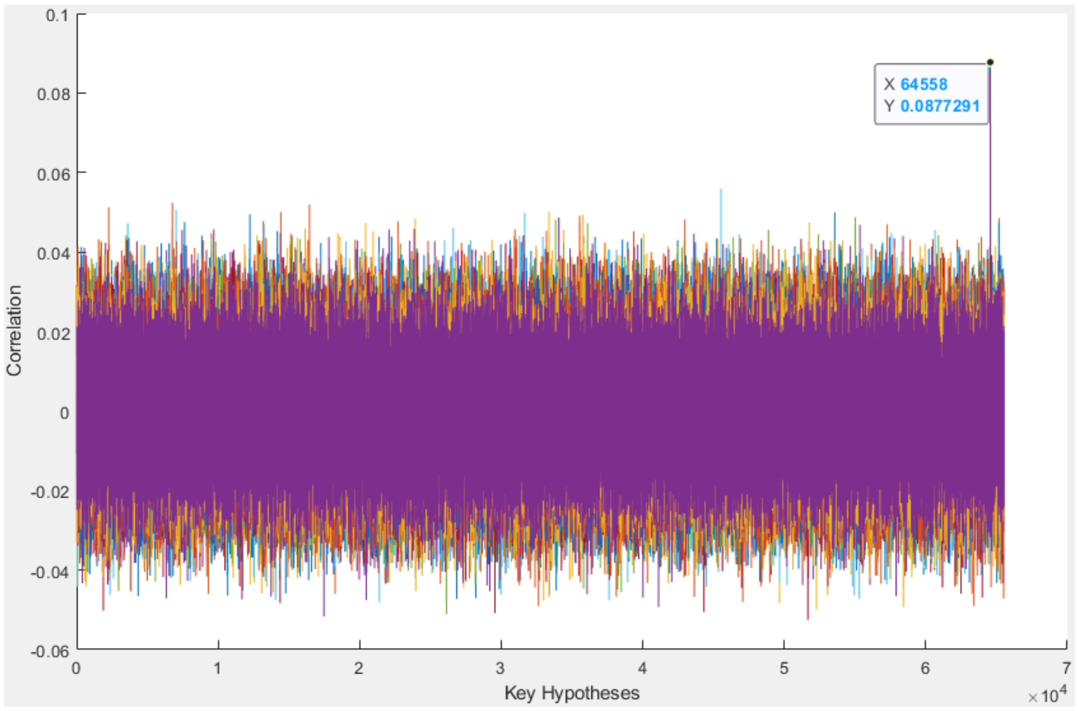


Figure 8.7: Correlation – Key Hypotheses Plot

9 Conclusions

In this thesis we focused on AES and its operations. We noticed that although it is a block cipher algorithm and splits data into blocks of 16, 28 or 32 bytes (depending on the key length), it processes these blocks byte after byte. Consequently, the instantaneous consumption of a device running AES depends on single bytes.

By using power models, we were able to simulate hypothetical power consumptions for all possible key values. Afterwards, we utilized the Pearson Correlation Coefficient (ρ) in order to compare the hypothetical consumptions with the measured power traces. The benefit using ρ, is that allows us to determine the relationship between two variables using different metric system. Note that our hypothetical consumption is simulated with the HW model. Hence, this matrix contains values from 0-8 (essentially, the HW counts the bits set to 1 in a byte). On the other hand, the power traces measure relative power consumption.

We showed that an unprotected AES implementation on an embedded device can be broken relatively easy. In our case, just 100 traces were enough to reveal the key byte under attack.

As a countermeasure, we explained what masking is, its types and how each one works. In any case, the goal of masking is to remove the dependency between the plaintext byte being processed and the power consumption. As a proof of concept, we mounted a 1st Order CPA attacks against the masked AES.

During our attack, the secret key remained well hidden against our typical 1st Order CPA, even after using 10.000 power traces. So, we needed to add more traces to be certain. Since our computed resources are limited, an incremental calculation of ρ was required, as described in [4]. Thus, we added more power traces while being able, at the same time, to keep the resources usage low. We recorder as many as 5.000.000 traces. Afterwards, we applied the incremental attack using all available traces and the key was, once again, not recovered. At this point, we concluded that masking actually does what it promises.

As we moved deeper into the CPA attacks, we explored 2nd Order Attacks in the scientific literature. The theory behind these attacks was explained and we attempted to mount one. The results were positive, as the key was recovered after approximately 5.500 power traces.

However, what troubled us was the amount of computations needed for a 2nd order attack to succeed. As the key hypotheses are now 65.536, the resources required exceed the memory capacity of a typical computer system. Hence, windowing needs to be applied, in order to break down the attack into smaller, and more feasible, attacks on windowed traces. Undoubtedly, this solves the memory capacity problem, but the execution time remains high. Although, some approaches have been presented in literature, and we proposed our key-windowing technique, we believe that it is still an issue that needs to be investigated, in order to further improve efficiency in black-box environments, as well.

10 References

[1] Mangard, S., Oswald, E., & Popp, T. (2008). *Power analysis attacks: Revealing the secrets of smart cards* (Vol. 31). Springer Science & Business Media.

[2] Ding, A. A., Zhang, L., Fei, Y., & Luo, P. (2014, September). A statistical model for higher order DPA on masked devices. In *International Workshop on Cryptographic Hardware and Embedded Systems* (pp. 147-169). Springer, Berlin, Heidelberg.

[3] Fumaroli, G., Martinelli, A., Prouff, E., & Rivain, M. (2010, August). Affine masking against higher-order side channel analysis. In *International Workshop on Selected Areas in Cryptography* (pp. 262-280). Springer, Berlin, Heidelberg.

[4] Bottinelli, P., & Bos, J. W. (2017). Computational aspects of correlation power analysis. *Journal of Cryptographic Engineering*, *7*(3), 167-181.

[5] Durvaux, F., Standaert, F. X., Veyrat-Charvillon, N., Mairy, J. B., & Deville, Y. (2015, April). Efficient selection of time samples for higher-order DPA with projection pursuits. In *International Workshop on Constructive Side-Channel Analysis and Secure Design* (pp. 34-50). Springer, Cham.

[6] DeTrano, A., Guilley, S., Guo, X., Karimi, N., & Karri, R. (2015, June). Exploiting small leakages in masks to turn a second-order attack into a first-order attack. In *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy* (pp. 1-5).

[7] Coron, J. S., Rondepierre, F., & Zeitoun, R. (2018). High order masking of look-up tables with common shares. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 40-72.

[8] Bruneau, N., Guilley, S., Najm, Z., & Teglia, Y. (2018). Multivariate high-order attacks of shuffled tables recomputation. *Journal of Cryptology*, *31*(2), 351-393.

[9] Lu, J., Pan, J., & den Hartog, J. (2010, June). Principles on the security of AES against first and second-order differential power analysis. In *International Conference on Applied Cryptography and Network Security* (pp. 168-185). Springer, Berlin, Heidelberg.

[10] Gierlichs, B., Batina, L., Preneel, B., & Verbauwhede, I. (2010, March). Revisiting higher-order DPA attacks. In *Cryptographers' Track at the RSA Conference* (pp. 221-234). Springer, Berlin, Heidelberg.

[11] Li, W., & Yi, H. (2016). Second-Order Power Analysis Attacks against Precomputation based Masking Countermeasure. *International Journal of Smart Home*, *10*(3), 259-270.

[12] Reparaz, O., Gierlichs, B., & Verbauwhede, I. (2012, September). Selecting time samples for multivariate DPA attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems* (pp. 155-174). Springer, Berlin, Heidelberg.

[13] Belgarric, P., Bhasin, S., Bruneau, N., Danger, J. L., Debande, N., Guilley, S., ... & Rioul, O. (2013, November). Time-frequency analysis for second-order attacks. In *International Conference on Smart Card Research and Advanced Applications* (pp. 108-122). Springer, Cham.

[14] https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[15] https://en.wikipedia.org/wiki/Rijndael_S-box

[16] https://en.wikipedia.org/wiki/AES_key_schedule

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

[18] Kocher, P., Jaffe, J., & Jun, B. (1999, August). Differential power analysis. In *Annual international cryptology conference* (pp. 388-397). Springer, Berlin, Heidelberg.

[19] https://www.researchgate.net/figure/Simple-Power-Analysis-SPA-trace-showing-an-entire-Advanced-Encryption-Standard-AES_fig2_341513963

[20] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

[21] https://libguides.library.kent.edu/SPSS/PearsonCorr

[22] https://www.st.com/en/development-tools/sw4stm32.html

[23] http://www.asciitable.com/

[24] Coron, J. S., & Goubin, L. (2000, August). On boolean and arithmetic masking against differential power analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems* (pp. 231-237). Springer, Berlin, Heidelberg.

[25] Goubin, L. (2001, May). A sound method for switching between boolean and arithmetic masking. In *International Workshop on Cryptographic Hardware and Embedded Systems* (pp. 3-15). Springer, Berlin, Heidelberg.

[26] Chari, S., Jutla, C. S., Rao, J. R., & Rohatgi, P. (1999, August). Towards sound approaches to counteract power-analysis attacks. In *Annual International Cryptology Conference* (pp. 398-412). Springer, Berlin, Heidelberg.

[27] Messerges, T. S. (2000, August). Using second-order power analysis to attack DPA resistant software. In *International Workshop on Cryptographic Hardware and Embedded Systems* (pp. 238-251). Springer, Berlin, Heidelberg.

[28] https://github.com/CENSUS/masked-aes-c

[29] Waddle, J., & Wagner, D. (2004, August). Towards efficient second-order power analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems* (pp. 1-15). Springer, Berlin, Heidelberg.