

UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

**FEASIBILITY OF SIDE-CHANNEL ATTACKS ON
MICROCONTROLLERS & FPGAs**

Diploma Thesis

Tragoudaras Antonios

Supervisor: George Stamoulis

Volos, February 2022



UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

**FEASIBILITY OF SIDE-CHANNEL ATTACKS ON
MICROCONTROLLERS & FPGAs**

Diploma Thesis

Tragoudaras Antonios

Supervisor: George Stamoulis

Volos, February 2022



ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

**ΥΛΟΠΟΙΣΗΜΟΤΗΤΑ ΕΠΙΘΕΣΕΩΝ ΠΛΕΥΡΙΚΟΥ
ΚΑΝΑΛΙΟΥ ΣΕ ΜΙΚΡΟΕΛΕΧΤΕΣ & FPGAs**

Διπλωματική Εργασία

Αντώνιος Τραγουδάρας

Επιβλέπων: Γεώργιος Σταμούλης

Βόλος, Φεβρουάριος 2022

Approved by the Examination Committee:

Supervisor **George Stamoulis**

Professor,

Department of Electrical and Computer Engineering,

University of Thessaly

Member **Nestor Evmorfopoulos**

Associate Professor,

Department of Electrical and Computer Engineering,

University of Thessaly

Member **Nikolaos Moondanos**

Associate Professor,

Department of Electrical and Computer Engineering,

University of Thessaly

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Georges Stamoulis for allowing me to work on a captivating topic and for playing a crucial role in shaping my academic interests during my undergraduate studies. It was his careful guidance and proficiency that enabled me to successfully carry out the challenging tasks incorporated in completing this thesis.

Last but not least, I would like to thank my beloved family members and friends for their unconditional love and support all these years.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Tragoudaras Antonios

Abstract

All electronic devices are prone to unintentionally reveal information pertinent to the underlying operations and real-time data processing, taking place in their processing cores through indirect sources, to some extent. The most common sources, mentioned as side-channels, are power consumption, electromagnetic emissions, computations timing. Side-channel attacks sample the information leaked through these inherent sources/channels, aiming at the interpretation of the collected data in order to extract invaluable information, critical for undermining the security of the device under attack.

This Thesis employs two of the most dominant form of Side-Channel attacks, namely Power Analysis and Fault Injection, against cryptographic schemes implemented to be used both for FPGA and microcontroller targets respectively. Issues arising from the feasibility and efficacy of the aforementioned attack techniques are addressed by proposing countermeasures for our own demonstrated attacks when possible.

Περίληψη

Όλες οι ηλεκτρονικές συσκευές είναι επιρρεπείς στην ακούσια αποκάλυψη πληροφορίας συσχετιζόμενη με τις υποτρέχουσες λειτουργίες και την επεξεργασία δεδομένων σε πραγματικό χρόνο, που πραγματοποιούνται στους πυρήνες επεξεργασίας τους διαμέσου πλευρικών πηγών, ως ένα βαθμό. Οι πιο συνηθισμένες πηγές, που συχνά αναφέρονται και ως πλευρικά κανάλια, είναι οι εξής: κατανάλωση ενέργειας, ηλεκτρομαγνητικές εκπομπές, χρονική διάρκεια υπολογισμών. Οι επιθέσεις πλευρικού καναλιού δειγματοληπτούν τη πληροφορία που διαφεύγει διαμέσου των εγγενών αυτών πηγών/καναλιών, με στόχο την διερμηνία των συλλεχθέντων δεδομένων, προκειμένου να εξαχθεί κρίσιμα πληροφορία που θα χρησιμοποιηθεί στη συνέχεια για την υπονόμευση της ασφάλειας της συσκευής υπό επίθεση.

Αυτή η διπλωματική κάνει χρήση των δύο πιο κυρίαρχων μορφών πλευρικών επιθέσεων, πιο συγκεκριμένα την ανάλυση ισχύος & έγχυση σφάλματος, ενάντια σε αλγορίθμους κρυπτογράφησης, σχεδιασμένοι να χρησιμοποιηθούν σε στόχους υλοποιημένους για FPGA και μικροελεγκτές. Τα ζητήματα που προκύπτουν από την αποτελεσματικότητα και υλοποιησιμότητα των ανωτέρων τεχνικών επίθεσης, αντιμετωπίζονται με την πρόταση αντίμετρων πάνω στις δικές μας επιθέσεις επίδειξης, όπου αυτό κατέστη δυνατό.

Table of contents

Acknowledgements	ix
Abstract	xiii
Περίληψη	xv
Table of contents	xvii
List of figures	xix
List of tables	xxi
Abbreviations	xxiii
1 Introduction	1
1.1 Thesis Objective & Contribution	2
1.2 Thesis Outline	2
2 Background in Side-Channel-Attacks(SCAs)	5
2.1 Power Analysis	7
2.1.1 Differential Power Analysis (DPA)	8
2.1.2 Correlation Power Analysis (CPA)	11
2.2 Fault Injection	12
2.2.1 Single Bit Fault Analysis	14
2.2.2 Differential Fault Analysis	16
3 Power Analysis Attacks	21

3.1	Power Analysis attacks against software-based implementations of cryptographic schemes	22
3.1.1	DPA against AES block cipher on XMEGA uC	23
3.1.2	CPA against AES block cipher on XMEGA uC	29
3.1.3	Bootloader AES-256	31
3.2	Power Analysis attacks against hardware-based implementation of cryptographic schemes	34
3.2.1	CPA against AES block cipher on Artix7	35
3.2.2	<i>Whitebox</i> Attack against Elliptic Curve Cryptography	41
3.3	Countermeasures	43
4	Fault Injection Attacks	45
4.1	DFA attack using phoenixAES/Clock glitching settings	46
4.2	Shortcomings of clock glitching / Countermeasures	50
5	Conclusion and Future Work	51
5.1	Conclusion & Contributions	51
5.2	Future Work	52
	Bibliography	53
	ChipWhisperer Platform/Tools	59
1	Artix FPGA board w/ Capture Lite board	59
2	XMEGA microcontroller w/ Capture Lite board	60

List of figures

2.1	Evolution of SCAs (Figure from [1])	5
2.2	Categorization of SCAs as illustrated in [1]	6
2.3	Attack on a single key-byte, the distinct spikes occurs at the time event(namely the first S-Box transformation) where the microprocessor is fusing key bytes with intermediate state bytes together. The green spikes represent the correct guess for second key byte, whereas the red ones represent a random-false key-guess	10
2.4	Demonstration of linearity connection between average current and Hamming Weight of leakage model's result C , of a single-byte data manipulation occurring during the S-Box transformation of the first AES-128 round. ChipWhisperer-Lite Capture board was used during the power measurements	12
2.5	Fault attack illustrated	15
2.6	Single bit fault attack injected in the last round key-addition, namely XOR operation between intermediate state 10th round-key bits	16
2.7	AES encryption structure(picture's source: [2])	17
3.1	A collected power trace illustrating the 'signature' of 10 AES rounds.	23
3.2	Using ChipWhisperer Analyzer for analysis phase, which utilizes Pandas DataFrames.	31
3.3	Difference between the average of all power traces and a single different power trace at a time, each illustrated with a different color to represent a different possible signature-value. The large red peak represents the difference between the mean of all power traces and the correct key guess of the first signature byte. The peak occurs at the point in time where the signature check takes place.	33

3.4	Difference between the average of all power traces and a single different power trace at a time, each illustrated with a different color to represent a different possible signature-value.	35
3.5	High-Level illustration of the targeted hardware AES implementation(figure's source: CW305 Whitepaper)	36
3.6	Utilize LASCAR python module to diminish CPA calculation time.	38
3.7	Plotting the best correlation achieved for all single key-byte guesses at a given time/sample.	39
3.8	PGE over the number of power traces used so far.	40
3.9	Colored waves represent the best key guess, black waves illustrate the second best key guess.	40
4.1	Block diagram of glitching 'sub-module' of Capture Lite Board(picture's source: ChipWhisperer ReadtheDocs)	45
4.2	Capture all ten rounds of AES to visualize the point where fault need to be injected.	47
4.3	The red vertical line illustrates the 13000th sample around which the glitches are inserted.	48
4.4	Glitch results of our 'campaign-attack' after sweeping through the glitch-parameters values provided in the text.	49

List of tables

- 2.1 Meaning of symbols used in Pearson Correlation Coefficient equation 2.2 12
- 2.2 Meaning of symbols in detailed Pearson Correlation Coefficient calculations 2.3 (taken from [1]) 13
- 3.1 Encrypted data format sent to the bootloader through a serial port with baud rate of 38400. 32
- 3.2 Incorrect guesses for a given distinguisher as the power traces acquisition is decreasing 43

Abbreviations

SCA	Side-Channel Attack
AES	Advanced Encryption Standard
ECC	Elliptic Curve Cryptography
RSA	Rivest–Shamir–Adleman public-key cryptosystem
FPGA	Field Programmable Gate Array
uC	Microcontroller
IC	Integrated Circuit
VLSI	Very Large - Scale Integration
SAD	Sum of Absolute Differences
SPA	Simple Power Analysis
CPA	Correlation Power Analysis
DPA	Differential Power Analysis
DFA	Differential Fault Analysis
CW	ChipWhisperer
IoT	Internet of Things
CPU	Central Processing Unit
ARM	ARM Ltd.
EM	Electromagnetic
S-Box	Substitution Box
LSB	Least significant Bit
AC	Alternating current
ADC	Analog-to-digital converter
LNA	Low Noise Amplifier
PGE	Partial Guessing Entropy
CBC	Cipher Block Chaining

IV	Initialization Vector
ECB	Electronic Code Book
LASCAR	Ledger's Advanced Side-Channel Analysis Repository
PCB	Printed Circuit Board
TVLA	Test Vector Leakage Assessment
PnR	Place and Route
DoM	Difference of Means

Chapter 1

Introduction

As modern computing systems tend to get faster due to breakthroughs made in the broader discipline of Computer Engineering during the course of the last decades, cyber attacks are considered the despairs of digital era compromising data confidentiality and resulting in financial losses. Software-based, Zero Day attack implications can be mitigated with security patches and software updates; however resolving attack impacts, which exploit vulnerabilities entrenched in hardware is extremely difficult if not a doomed task, since there is no equivalent to a security-fix hardware-wise, meaning that the effects of such Zero Day attacks will be propagated across many device for an extended period in time, or even a full 'hardware-lifecycle'. During the first quarter of 2018, both Google Project Zero team and individual security analysts reported Spectre [3] and Meltdown [4], two vulnerabilities affecting several modern CPUs and uCs. Spectre refers to a broad family of potential weaknesses of which Meltdown is one. Both exploit CPU's architecture performance optimizations techniques, namely speculative execution/evaluation and out-of-order execution, resulting in jeopardizing security implications, ranging from violating process isolation boundaries to exfiltrate data from kernel memory space or even breach browser sandboxing. This was just a reminder of how far side-channel attacks have come since originally introduced in the mid '90s [5], followed by works leveraging physical information, such as power consumption [6] and electromagnetic emissions [7] the next few years.

The immense usage of ICs in consumer electronics aligned with the continuous growth of IoT market and their application realm, create back-doors for larger attacks that will not only influence the operation of a single device rather than a greater interconnected network [8].

It would be reasonable for one to suggest that state of the art electronic devices would have some level of protection against attacks that have been publicly disclosed for almost 23 years. Well the latter is far from truth since variants of the aforementioned side channel attacks are still very powerful to this day.

1.1 Thesis Objective & Contribution

This work intends to demonstrate the capabilities of power analysis and fault injection techniques to recover sensitive data, typically that being secret keys, used during the encryption/decryption operations in otherwise impenetrable well-known cryptographic schemes, running both on modern uC and FPGAs used in a variety of IoT applications. The intimate knowledge obtained from our experiments, allows us to propose alterations on these crypto-routines implementation-wise, in order to impede such attacks from undermining the security of modern consumer electronics powered by similar ICs as the ones used in this thesis. All the tools used in this study are open-source, low-cost and part of the ChipWhisperer open-source project. This thesis can also be considered as a survey of Power Analysis & Fault Injection attacks and their existing countermeasures.

1.2 Thesis Outline

The rest of this Thesis is organized in the following Chapters:

- Chapter 2: An in-depth dive in Side-Channel Attacks, where we focus on demystifying the theoretical framework behind Power Analysis & Fault Injection attacks. The objective of this chapter is to provide profound knowledge needed for a comprehensive understanding of our hands-on experiments, carried-out in Chapters 3&4.
- Chapter 3: Demonstration of power analysis attacks against different crypto-routine&target combinations.
- Chapter 4: Exhibition of fault injection attacks against different crypto-routine&target combinations.

- Chapter 5: Conclusion of our work and its findings/contributions; proposal for future/prospective work as an extension of this thesis.

Chapter 2

Background in

Side-Channel-Attacks(SCAs)

Undeniably, P. Kocher's works [5, 6] were groundbreaking and had a seminal influence to hardware security research field, however the first ever mention to a Side-Channel Attack dates back to mid 60's. Back then, British Intelligence Agency utilized the sound produced by a cryptographic rotor-type machine to fully derive the secret key [9]. As illustrated in 2.1 modern attacks have evolved over time, refining the way information inherent in physical channels is utilized.

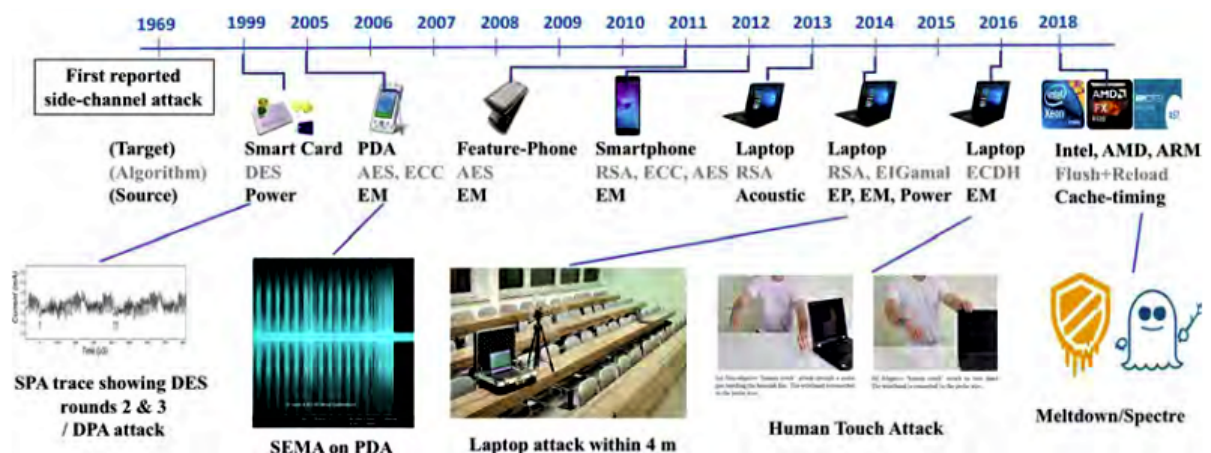


Figure 2.1: Evolution of SCAs (Figure from [1])

Side-Channel attacks can be divided into two major categories :

- **Passive:** The functionality of the targeted device is not interfered in any way by the attacker party. We just assume that the attacker has access to the device under attack and

is able to monitor some kind of physical information generated under normal operating circumstances. In the context of cryptographic implementations on such devices, the implicit presence of intermediate values (i.e. intermediate computation results, fusing plaintext and key together), in the form of Side-Channel *leakage* (i.e. a power consumption, EM radiation), typically needed to be computed during the ciphertext generation, can provide enough '*clue*' for the secret key to be extracted.

- Active: This kind of attacks require the attacker to be able to explicitly alter the standard operation of the '*prey*' device on demand. By doing so the attacker usually will be able to force functional errors influencing the '*final*' computation outcome (ciphertext in cryptographic context), ultimately leading to a very specific Side-Channel leakage that will allow him/her to obtain confidential '*intelligence*'. Such attacks require proficient knowledge related to the computations which take place during the device operation and cannot be performed using a black-box approach.

The figure below 2.2 groups the attacks into dedicated categories based on the origin of the physical channel information, namely: Power, EM, timing, Fault Injection. Each of the previous SCAs classified under the *Passive Attacks* category can be further divided according to either the statistical analysis/methods applied to the acquired physical measurements (in case of Power and EM analysis) or how the attack was driven in the first place (in case of timing analysis). Fault injection attacks typically get splitted using the physical component (clock pins i.e.), which gets disturbed to create faulty behaviour, as a reference.

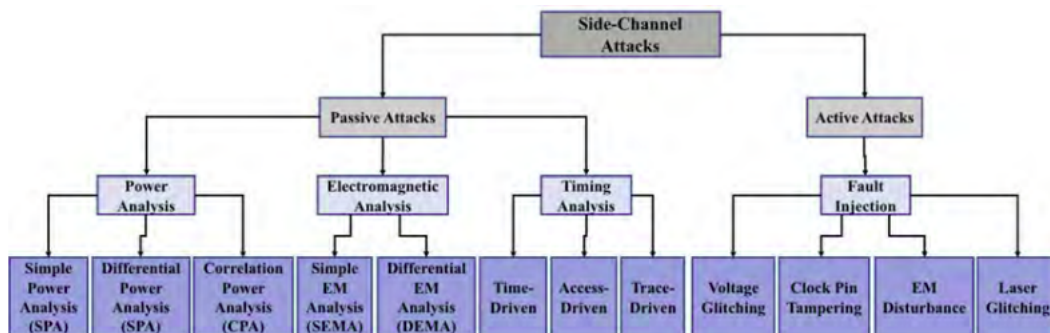


Figure 2.2: Categorization of SCAs as illustrated in [1]

Our work focuses both on Power Analysis and Fault Injection attacks mainly against symmetric cryptography, but there is also a small section where public-key cryptography is addressed. Fault Injection attacks fitting in the semi-invasive category are out of the scope of this Thesis

(i.e. physical probing and other attacks requiring 'on-chip' modifications [10, 11]), as only clock & voltage glitching techniques are employed. Also for the rest of this study we assume the reader to be familiar with Advanced Encryption Standard (AES) and Elliptic Curve Cryptography (ECC), especially with AES as it is extensively used throughout our experiments in Chapter3 & Chapter4 and is the *all-time favorite victim* of SCAs.

2.1 Power Analysis

Power Analysis attacks are considered amongst many researches a very dominant form of Side-Channel attacks if not the most, while they have been utilized utterly since their original introduction by no other than P.Kocher in 1999 [6]. This type of attacks are well known for their effectiveness in breaking both hardware and software implementations of cryptographic schemes/routines. For instance, the AES key can be derived easily in a few minutes time by applying such power analysis methods. The three most known methods are depicted in figure 2.2 under the power analysis *branch*.

Independently of which analysis method is used an attacker needs to monitor the power consumption of the target device¹, during encryption/decryption operations, prior to applying some type of statistical analysis to the collected traces². The latter action is critical, since the power consumed by an IC is directly dependent on the operations taking place on this digital electronic. The very same principal was exploited in [6] by applying a SPA to successfully break the RSA cryptography in late 90's. In this Thesis we mostly use CPA & DPA as these types of analysis are a great match for breaking AES 'variants' implementations for FPGA and uC targets(referred earlier as hardware and software respectively), exhaustively examined in Chapter3.

¹The acquisition of power signals in this study is done through the utilization of ChipWhisperer-Lite Capture board. The process of collecting power traces for the sake of our experiments is discussed concisely in Chapter3

²A power trace is defined as the recorded power consumption of the our 'victim' device, starting at the beginning of a(n) encryption/decryption 'run' and lasting until the very end of the same 'run'. Usually every cryptographic algorithm is known to have a distinctive *power signature*, revealing sensitive information such discrete operation executed by our target in 'real-time'.

2.1.1 Differential Power Analysis (DPA)

Although SPA utilizes the information leakage related to the type of computations executed during digital electronic operation, through the physical power channel, its shortcoming lies to the fact that this property alone is not effectively enough to break an AES implementation. A more powerful approach was also initiated in [6], where the authors claimed that power side channels encapsulate/enclose information about the data (secret keys & plaintexts) 'manipulated' on the target device, rather than merely the undergoing discrete cryptographic operations. This is mainly due to the dynamic power consumed by an IC caused from the switching activity of the transistors and their capacitive behavior. The author of [12] illustrated the latter in a very pellucid manner:

"A data bus on a digital device is driven high or low to transmit signals between nodes. The bus line can be modeled as a capacitor, and we can see that changing the voltage (state) of a digital bus line takes some physical amount of energy, as it effectively involves changing the charge on a capacitor"[12].

"Microcontrollers set their internal buses to a constant state before the final value is loaded"[12].

DPA takes full advantage of the aforementioned power dependency on sensitive data, namely the cryptographic-key, while it is applicable in a variety of devices running the AES [13].

In this paragraph we explain our attack model for DPA based on the original attack [6] in detail, so we do not need to repeat ourselves later in Chapter 3. Assume an attack point in time, where an intermediate operation f fuses/mixes known input P (plaintext) with secret data K (key) together, where

$$C = f(P, K) \quad (2.1)$$

is the net result of this operation. Due to power consumption implicit correlation to the computed data on the device, it is anticipated that this will also be noticeable on the recorded power traces. Then we proceed to capture N traces of equivalent AES complete encryption/decryption runs, where a fixed unknown K is used and some random input P is given every run. If we achieve to generate $C' = f(P, K')$ where K' is representing all possible key values of the actual K , while K used during the initial power trace acquisition, it is feasible to make a comparison between our recorder power trace and the hypothetical/generated ones in order to determine the best key guess. The following properties are hard-requirements for the success of our DPA model:

- It is crucial for f to depend only on a single byte of the actual key K , as this defines a computational feasible search key-space of 2^8 . In order to recover the full key one should just repeat the attack for each byte of the 16-byte key in case of AES-128 variant.
- f , which typically is called leakage model by the research community, ideally has to be a non-linear function. For that reason, plus the single key-byte dependency requirement mentioned in the 'bullet' above, the S-Box transformation of the first/last AES round is considered a very effective choice. Even a guess K' with only a single bit fault (compared to K) will result in multiple "wrong bit" values at the output of S-Box C' , as S-Box transformation perfectly meets the non-linearity properties needed. As a result this will allow the attacker to readily distinguish the golden/best key guess during his comparisons.

Yet we have not discussed a convenient way/metric to compare our hypothetical leakage C' to the actual leakage C (which is implicitly embedded into our recorded power traces). Most works adopted a technique originally proposed in [6] or at least some alteration in it. So the fundamental principal behind most DPAs is divided in the following delineated steps:

1. Group the acquired power traces based on a single bit of the targeted key 'sub-byte' (usually the LSB) of C' for all possible key-combinations.
2. Calculate the (Absolute) Difference of means between the two groups generated in the previous step, for all possible C' (and implicitly for all different K').
3. For all incorrect guesses we anticipate a low and similar difference of means, since the groups have been classified randomly. On the other hand if our guess is correct we should see a point in time where there is a large spike. This spike occurs due to power traces have been correctly divided in groups where the hypothetical leakage matched with the actual power leakage.
4. The point in time where the spike occurs corresponds to the time-event of data getting manipulated by our leakage model f .

In our experiments we have followed these steps scrupulously. One known artifact of this approach is retrieving the largest 'part' of the key successfully but not all (resulting in 1 or 2 wrong/faulty bytes). This problem is typically referred as ghost peaks (which lead to the

wrong sub-key guesses) and has been tackled ingeniously in [14, 15]. However this requires a lot of rethinking and in order to get rid of ghost peaks we simply make a small alteration in steps 1 & 2 of the 'conventional' approach, previously used in our attacks against AES. Instead of grouping the traces once, based on a single bit (LSB) of the targeted subkey-byte we simply repeat this for all bits of the targeted byte, leaving us with 8 sets(each consisting of two groups¹). Then we need to calculate the difference of means across all groups of all sets and base our decision for the best key-guess on the average difference of means of all sets, essentially taking into account the contribution of all 8-bits in power consumption, instead of single, in a single byte attack scenario. This will lead to always recover the full-key correctly. Figure 2.3 illustrates an attack against the second key sub-byte of AES-128(running on XMEGA uC, more on Chapter 3) with our 'refined' DPA approach.

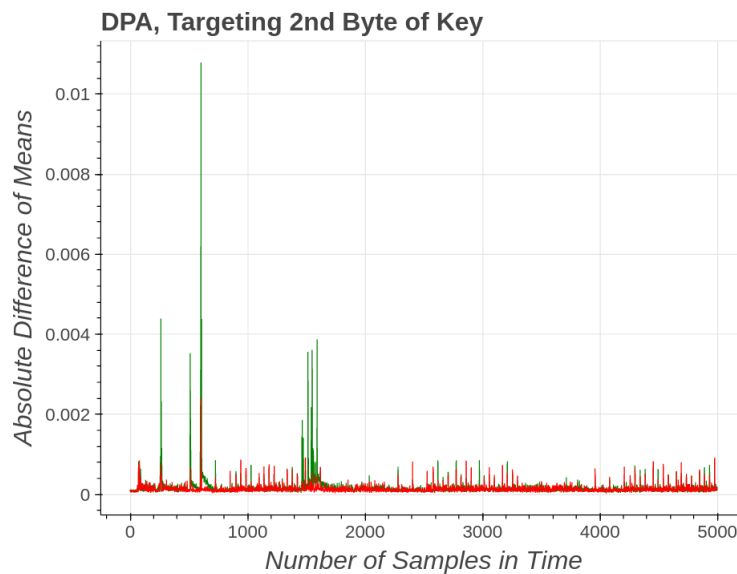


Figure 2.3: Attack on a single key-byte, the distinct spikes occurs at the time event(namely the first S-Box transformation) where the microprocessor is fusing key bytes with intermediate state bytes together. The green spikes represent the correct guess for second key byte, whereas the red ones represent a random-false key-guess

¹the groups have the same property as in the 'conventional approach'

2.1.2 Correlation Power Analysis (CPA)

Usually DPAs require a large amount of power traces to be collected, while the 'refined' approach discussed earlier needs to spend almost $\times 7$ more minutes¹ for the statistical analysis calculations² phase when compared to the conventional approach. A more powerful Power Analysis attack against AES implementations is CPA, proposed in 2004 [16]. CPA's attack model exploits the correlation between power and data set in the bus, namely intermediate data computations, more effectively by default. Usually such attacks need far less power traces than DPA, while the statistical analysis computations^{2.2} require a considerable smaller time to be completed, resulting in fully deriving a cryptographic key in less than a minute or even in seconds in some of our experiment cases (Chapter 3).

CPA is based on the assumption that there is a linear relationship between the power consumption and the Hamming Weight³ of leakage model's result, namely C in 2.1. In other words CPA models account for the contribution of each bit of the leakage model f result to the power consumption inherently. In Figure 2.4 this linear connection of Hamming Weight and the power consumption is validated for an AES implementation running on XMEGA uC, extensively used in our experiments.

The CPA uses the Pearson Correlation Coefficient 2.2 to compare the relationship between the Hamming Weight hypothetical leakage of C' denoted as Y and the recorder power traces denoted as X . A wrong key guess K' will result in low coefficient values, whereas our best guess for the key is the single-byte value of K' with a sharp peak in the calculated values^{2.3} at some point in time, where the S-Box transformation/computation is taking place. Since we attack a single byte of the key at a time we can recover the full cryptographic key, as this is again a tractable problem of search-space 2^8 needed to be repeated 16 times.

$$r = \frac{cov(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sqrt{E[(X - \mu_X)^2(Y - \mu_Y)^2]}} \quad (2.2)$$

¹Conventional DPA: 5 minutes, Refined DPA 3.1.1: 35 minutes (Calculations performed on a typical x86 laptop processor)

²Difference of Means

³The number of bits set to one in a single byte in our case

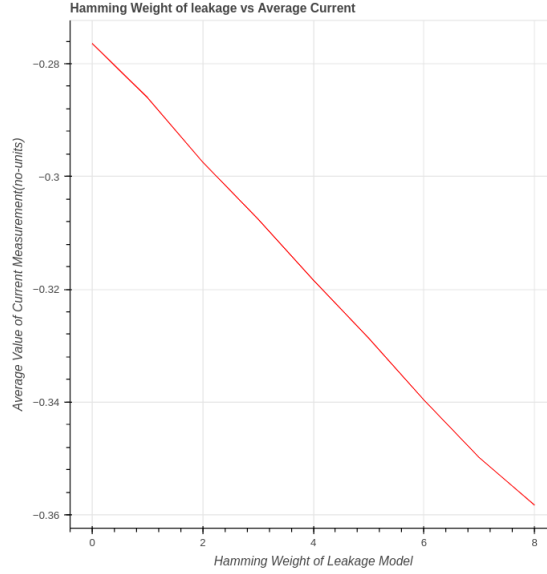


Figure 2.4: Demonstration of linearity connection between average current and Hamming Weight of leakage model's result C , of a single-byte data manipulation occurring during the S-Box transformation of the first AES-128 round. ChipWhisperer-Lite Capture board was used during the power measurements

Symbol	Meaning
$cov(X, Y)$	covariance between X and Y
σ_X	standard deviation of X
σ_Y	standard deviation of Y
E	Expectation
μ_X	mean of X
μ_Y	mean of Y

Table 2.1: Meaning of symbols used in Pearson Correlation Coefficient equation 2.2

$$r_{i,j} = \frac{\sum_{d=1}^D (t_{d,j} - \bar{T}_j)(h_{d,i} - \bar{H}_i)}{\sqrt{(\sum_{d=1}^D t_{d,j} - \bar{T}_j)^2 (\sum_{d=1}^D h_{d,i} - \bar{H}_i)^2}} \quad (2.3)$$

2.2 Fault Injection

Fault attacks were first conceived in the late 90's [17] almost aligned with the emergence of Power Analysis attacks. The authors of [17] built theoretical methods that could pose a serious threat for RSA and other cryptographic schemes at the time. Since then, these methods

Equation	Value
d	current trace
D	total number of trace collected
i	key guess
j	sample point in time
h	Hamming-Weight of hypothetical leakage C'
\bar{H}	Mean of all Hamming Weight values for a given key guess
t	a single power trace
\bar{T}	Mean of power traces values for a given sample point in time

Table 2.2: Meaning of symbols in detailed Pearson Correlation Coefficient calculations (taken from [1])

were able to be tested practically against AES[18], DES[19], RSA[20] running on custom IC explicitly/specifically designed for facilitating fault injection attacks. The [21] is a survey that includes fault injection attacks against AES and RSA.

In contrast to Power Analysis attack techniques, fault injection attacks can be utilized to target a wider range of application, running on embedded systems, rather than only cryptographic cores/routines. For instance in [22] voltage glitching via USB allowed a hardware engineer to dump a firmware image out of custom IC. In general, fault injection attacks cause electronic devices to behave unexpectedly, which includes tampering computation results, programs skipping instructions, corrupting the data saved in memory regions etc. The two most common forms of fault injection attacks are clock & voltage glitching.

- **Clock Glitching:** These attacks try to generate a noisy/glitchy sharp edge near falling/rising edges in the normal-reliable clock signal fed to the device under attack. The latter creates setup/hold violation ultimately allowing the attacker party to apply some type of 'post-analysis' in the faulty 'output' of a system to fulfil the attack's purpose/ motivation.
- **Voltage Glitching:** Electronic devices operate in specific voltage input levels, if for some reason these levels are not met during their operation malfunctions occur. By being able to disrupt the voltage supply to an operating hardware chip by 'under-

voltaging' it, this is certain to cause faults in turn leading to a chain-reaction of events which propagate these fault across all device's computations. Again such fault can avail attackers in terms of undermining the security of a device by revealing information that can be exploited in the same way as with clock glitching 'post-analysis'.

Our work focuses mainly on two clock glitching attacks¹ both targeting the AES block cipher:

- A software AES implementation running on XMEGA uC
- A hardware AES implementation running on Artix-35T, Xilinx's 7-series FPGA

Avid readers are referred to our Github repo,² where apart from clock glitching attacks we also examine voltage glitching for bypassing password checks & dumping firmware out of memory (for a simplified systems).

In the following subsections we examine the attack models used after successfully injecting a voltage/clock glitch in order to recover the secret key of AES. Specifically the logic behind Differential Fault Analysis, which is demystified in 2.2.2, is of great importance as it allowed us to break the software & hardware implementations mentioned earlier.

2.2.1 Single Bit Fault Analysis

As mentioned in the beginning of this chapter, fault injection attacks require thorough knowledge of the design that is about to be attacked as black box approaches is not applicable in these cases. The success of fault injection attacks is highly dependent on attackers ability to inject fault with precise timing at the right moment, especially in our work where we study attacks against AES block cipher.

In order to understand the most practical & applicable DFA2.2.2, we firstly need to build a solid understanding of how simpler Fault Analysis attack work. The basic idea behind fault

¹The generation of clock glitches in this work is done through the ChipWhisperer-Lite Capture board. The process of generating & injecting clock glitches in our targets for the sake of our experiments is briefly discussed in Chapter4

²Note that this is a forked repository. Our experiments were greatly influenced by the knowledge obtained from the open-source ChipWhisperer Project

attacks is illustrated in Figure 2.5. The first assumption made is that the attacker party has the capability of sending messages (plaintext) to be encrypted by the desired device under attack, while being able to collect the encrypted results (ciphertexts). If the attacker is able to induce faults that influence the encryption result's integrity this would lead to a so called faulty ciphertext. So the objective here for the attacker is to find a way to exploit the difference between a faulty and a faulty-free encrypted message, both generated with an identical plaintext, which will allow him/her to obtain the secret key used during the encryption. As stated before the fault generation is discussed in Chapter 4.

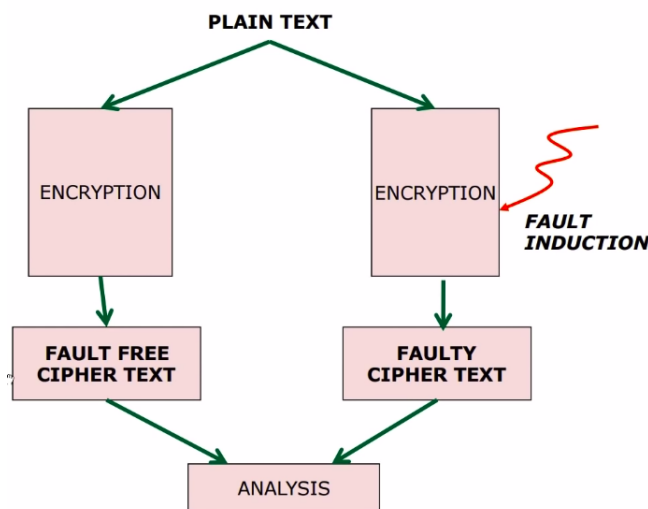


Figure 2.5: Fault attack illustrated

A single bit fault attack, otherwise known as stuck-at-zero fault, against the *AddRoundKey* operation of the last round of AES is considered in Figure 2.6. In a normal full-run of AES block cipher C represents the final generated ciphertext, where c_0 is the result of $0th$ bit XOR operation $s_0^1 \oplus k_0$. Let's consider the case where the attacker is able to inject a stuck-at-zero fault in the s_0 , this will result in k_0 always getting propagated to the ciphertext bit, which is the single-bit \oplus result, essentially meaning that $c_0 = k_0$. By repeating this stuck-at-zero fault injection attack for all 128-bit separately (s_0, s_1, \dots, s_{127}) at \oplus last-round key-addition, the 16-byte value of 10-th round key will be trivially obtained. Since AES key_schedule is publicly disclosed the attacker can recover the $0th$ key round, by inverting the key-expansion operations, which is identical to the AES secret key.

¹0th Intermediate state bit before last round's key-addition

The shortcoming of this fault analysis method to be practical and applicable in real applications, is the fact the injecting a fault precisely at the time where the last AddRoundKey is taking place, while also 'manipulating' exactly one bit is extremely difficult as it requires precise fault induction 'control', while fault attacks affecting a single bit only are considered infeasible.

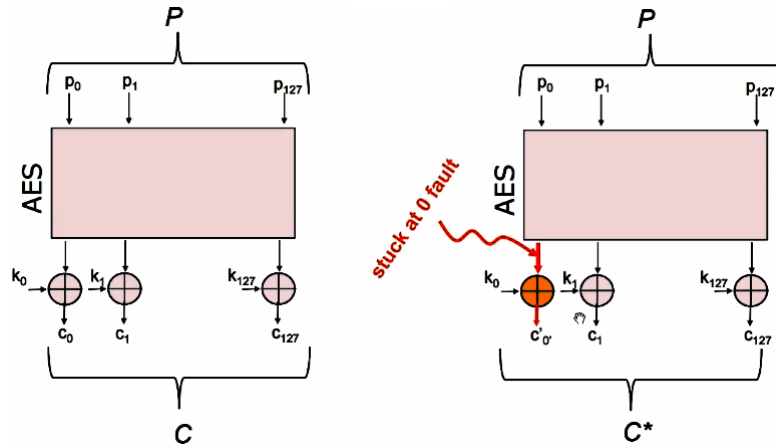


Figure 2.6: Single bit fault attack injected in the last round key-addition, namely XOR operation between intermediate state 10th round-key bits

2.2.2 Differential Fault Analysis

Since the simple fault analysis model discussed above is quite an impracticable approach, several works tried to reduce the number of faults required to recover the key of a cryptographic scheme, while also relaxing the constraints for the fault model, namely the fault injection attacks can affect several bytes over a larger scope of encryption rounds instead of a very specific bit manipulation at a very particular operation (like in stuck-at-zero fault model).

In this subsection, the fault model of choice used in the fault analysis part of our experiments is explained adequately. Our work adopts the Differential Fault Analysis initially published in [18]. While this analysis is heavily based on mathematical formalism behind Galois Finite Fields and requires some level of familiarity with AES operations, we will try to convey a higher-level understanding/description of DFA. The Differential Fault Analysis on White-box AES Implementations¹ is an eloquent article for an in-depth examination of DFA.

¹As mentioned earlier fault injection attacks cannot be performed using a black-box approach.

The prerequisites for our attack model to work is the following:

1. We have to be able to repeatedly trigger complete 'runs' of the AES block cipher, with the same input/plaintext. The attack will still work for an unrevealed plaintext as long as it is constant throughout the encryption runs.
2. The output/ciphertext has to be monitored/saved in same way, since the difference between a faulty and a faulty-free ciphertext is crucial for the Differential Analysis.

All the attacker party needs to do now is to corrupt a single byte of the intermediate state anywhere between the last two MixColumns AES operation. This attack is also known as 9th Round MixColumns fault attack. As seen in Figure 2.7 the single byte fault attack can be injected in any of the following operation candidates:

- AddRoundKey \oplus with 8th key-round (K_8)
- SubBytes/S-Box 9th Round
- ShiftRows 9th Round

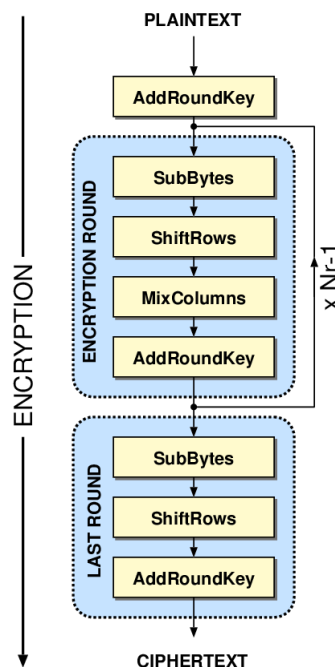


Figure 2.7: AES encryption structure (picture's source: [2])

One can easily realize that the fault model has already been relaxed noticeably since the only requirement for the fault is to be inserted between the time window of the last MixColumns

& after the penultimate MixColumns, while affecting a single byte of AES state.

Consider the AES state of a faulty free encryption run to be the following, just before the last

$$\text{MixColumns operation}^1 : \begin{bmatrix} S_0 & S_4 & S_8 & S_{12} \\ S_1 & S_5 & S_9 & S_{13} \\ S_2 & S_6 & S_{10} & S_{14} \\ S_3 & S_7 & S_{11} & S_{15} \end{bmatrix}$$

$$\text{While the fault-injected AES state exactly at the same time is: } \begin{bmatrix} X & S_4 & S_8 & S_{12} \\ S_1 & S_5 & S_9 & S_{13} \\ S_2 & S_6 & S_{10} & S_{14} \\ S_3 & S_7 & S_{11} & S_{15} \end{bmatrix}$$

The remaining operations needed to be applied in the states above until the generation of both the fault-free and the faulty ciphertext are the following :

1. 9th round *MixColumns*
2. AddRoundKey \oplus with 9th key-round (K_9)
3. SubBytes/S-Box **Last Round**
4. ShiftRows **Last Round**
5. AddRoundKey \oplus with 10th key-round (K_{10})

Since the MixColumns is the only operation that mixes multiple byte of the state, bytes of the same column to be precise, together it is anticipated that the faulty ciphertext will have four faulty bytes in the following state indices : $\{0, 7, 10, 13\}^2$.

Assume C_0 represents the 'zeroth' byte of fault-free ciphertext and C'_0 of the faulty one correspondingly. So we have the following equations (note that multiplication and additions(essential \oplus) here are done in $\mathbf{GF}(2^8)$) :

$$C_0 = \text{SBox}(2S_0 + 3S_1 + S_2 + S_3 + K_{9,0}) + K_{10,0} \quad (2.4)$$

¹The fault could also occur earlier, but for the simplicity of our demonstration we assume it occurs right before the last MixColumns operations

²The state-matrix is enumerated column-wise, while the reordering/spread of the four faulty bytes in all four columns, originally multiplied $\times 4$ due to MixColumns, arises after the ShiftRows operations.

$$C'_0 = SBox(2X + 3S_1 + S_1 + S_3 + K_{9,0}) + K_{10,0} \quad (2.5)$$

So if we combine together, which is the reason behind the analysis is called differential, by XORing¹ them: 2.4 \oplus 2.5 we get:

$$C_0 + C'_0 = SBox(2S_0 + 3S_1 + S_2 + S_3 + K_{9,0}) + K_{10,0} + SBox(2X + 3S_1 + S_2 + S_3 + K_{9,0}) + K_{10,0}$$

$$C_0 + C'_0 = SBox(2S_0 + 3S_1 + S_2 + S_3 + K_{9,0}) + SBox(2X + 3S_1 + S_2 + S_3 + K_{9,0})$$

$$C_0 + C'_0 = SBox(2S_0 + 3S_1 + S_2 + S_3 + K_{9,0}) + SBox(2X + \mathbf{2S_0} + \mathbf{2S_0} + 3S_1 + S_2 + S_3 + K_{9,0})$$

Assume:

$$Y_0 = 2S_0 + 3S_1 + S_2 + S_4 + K_{9,0}$$

$$Z = S_0 + X$$

Then we have:

$$C_0 + C'_0 = SBox(Y_0) + SBox(Y_0 + 2Z)$$

Similarly for the remaining affected bytes we can write:

$$C_7 + C'_7 = SBox(Y_1) + SBox(Y_1 + 3Z)$$

$$Y_1 = 3S_0 + S_1 + S_2 + 2S_3 + K_{9,3}$$

$$C_{10} + C'_{10} = SBox(Y_2) + SBox(Y_2 + Z)$$

$$Y_2 = S_0 + S_1 + 2S_2 + 3S_3 + K_{9,2}$$

$$C_{13} + C + 13' = SBox(Y_3) + SBox(Y_3 + Z)$$

$$Y_3 = S_0 + 2S_1 + 3S_2 + S_3 + K_{9,1}$$

Only a set of Z satisfy these equation for each $C_n + C'_n$ pair, while for each Y_n we can similarly find the following key candidates:

$$K_{10,0} = SBox(Y_0) + C_0$$

$$K_{10,7} = SBox(Y_1) + C_7$$

$$K_{10,10} = SBox(Y_2) + C_{10}$$

$$K_{10,13} = SBox(Y_3) + C_{13}$$

¹Addition in $\mathbf{GF}(2^8)$

It is proved that the system of equations above is easier to be solved with a quick brute force technique, by trying all possible values for Z & Y_n and using 'short-circuiting' to discard values as soon as they fail to meet any of the above equations. In our Github repo there is a proof of concept¹ of the latter in the form of Jupyter Notebook, where we prove that the injection of two successive random byte faults in the same column, following the 9th Round MixColumn model, is enough to determine four bytes of the 10th round key (i.e. with 2 successive byte faults at the first column we recover $\{K_{10,0}, K_{10,7}, K_{10,10}, K_{10,13}\}$). To recover the full-key we need to do this repeatedly for the remaining columns. In total 4×2 such successive faults are needed to successfully obtain the full K_{10} .

¹Note that this is a forked repository. Our work is an extension of the contributions to ChipWhisperer open-source repo. For completing the tutorial you do not need either a capture or a target board.

Chapter 3

Power Analysis Attacks

Before moving forward with the actual attacks, since we have already described in detail the fundamental principles behind DPA & CPA (Chapter2), we are going to deal with the power traces acquisition. Instead of using a high-end oscilloscope for this purpose, the progress made in embedded security field thanks to ChipWhisperer Project and his creator Colin O’Flynn, allowed us to work with low-cost open-source tools. ChipWhisperer-Lite Capture board has all the hardware components needed¹ for monitoring the connected device under attack. The capture board is compatible with a variety of ’victim’ devices², while pretty much any modern electronic powered by some type of processor unit, like a Raspberry Pi, can be modified to interface with the Capture board. Another shortcoming when collecting power traces to be used in Power Analysis attacks is that all measurements are in accordance to the oscilloscope’s internal clock. This requires usually a large number of data/traces to be collected also in a very high sampling rate for them to be exploitable in the power analysis phase. This problem is resolved with ChipWhisperer’s architecture [23], which supports synchronous sampling and clock recovery [24], allowing all our measurements to be in phase with target’s clock source³. The authors of [24] proved ChipWhisperer’s ’superiority’ even with reduced sample rate; much less power traces needed to be captured for achieving the same success rate metric⁴ of SCAs.

¹ADC-10bit at 105MS/s, AC-coupled analog-input with LNA, see ChipWhisperer-Lite Hardware Specifications.

²such as uC(AVR’s XMEGA(8-bit), ARM’s STM32F(32-bit)), FPGAs(Xilinx’s Artix7-35T). For all target boards see ChipWhiserer Victims.

³even if the target uses an internal oscillator, demonstrated also in Chapter 5 of [12].

⁴Average PGE (Partial Guessing Entropy).

All interactions both with capture & victim boards is possible through ChipWhisperer Software. The ChipWhisperer API is a detailed documentation of all available Python classes and methods providing invaluable help with both capture and analysis tasks, while taking care all serial communication between the user/attacker workstation and capture/victim boards.

3.1 Power Analysis attacks against software-based implementations of cryptographic schemes

For all our experiments described in the relevant subsections we need to acquire some power traces monitoring the device power consumption during complete runs of the targeted implementation, which can be explicitly triggered in demand, for arbitrary inputs (plaintexts), as mentioned in Chapter 2. The source code and all the binary files (.hex)¹ that are used to configure the XMEGA microcontroller are provided in ChipWhisperer Github repo. The communication with our 'victims' boards is facilitated by the SimpleSerial protocol adoption, while the programming of the XMEGA microcontroller is done utilizing the XMEGA PDI (Program and Debug Interface) programmer existent in ChipWhisperer-Lite capture board architecture. Describing in detail how to interact with the capture & and target board is pointless in the scope of this thesis, as related information can be found in ChipWhisperer comprehensive documentation.

In this section we just need to apply in practice the theoretical DPA & CPA models in 3.1.1 and 3.1.2 respectively for the targeted device, which is an AVR's 8-bit microcontroller. Both DPA & CPA attack the AES-128 software implementation of `avr-cryptolib`. However, in 3.1.3 we aim to the replication of the attack's results of [25], which is a more realistic attack against a bootloader², which sole purpose is the de-ciphering of 'trusted' encrypted firmware using the AES-256 CBC mode of operation, meaning that for a successful attack the attacker besides the 32-byte key needs to recover also an authentication signature and an Initialization Vector (IV)³. In the last part of 3.1.3 a countermeasure is proposed against the signature-

¹We rebuild our firmware in our experiments using the <https://www.gnu.org/software/make/> just to make sure that any changes in the source code are in effect, needed mostly for bootloader attack 3.1.3 proposed countermeasures.

²The source-code of bootloader is provided by the authors of [25] and is also available in the ChipWhisperer's Github repo.

³The reader of 3.1.3 should be familiar with AES CBC mode of operation.

recovery attack/method studied in [25].

3.1.1 DPA against AES block cipher on XMEGA uC

In this experiment we capture N power traces, each representing a complete AES-128 ECB run. We assume for each encryption individual run of total N runs to be fed with a random plaintext(input), while the encryption key of the AES is considered unknown and well secure¹. In Figure 3.1² the repetitive 'power behavior' of each AES round(a total of ten rounds) is easily distinguishable. This is due to the fact that round consists of the same transformations³(except for the last one). As discussed in Chapter 2 our leakage model f is selected to be the S-Box transformation of the first AES round.

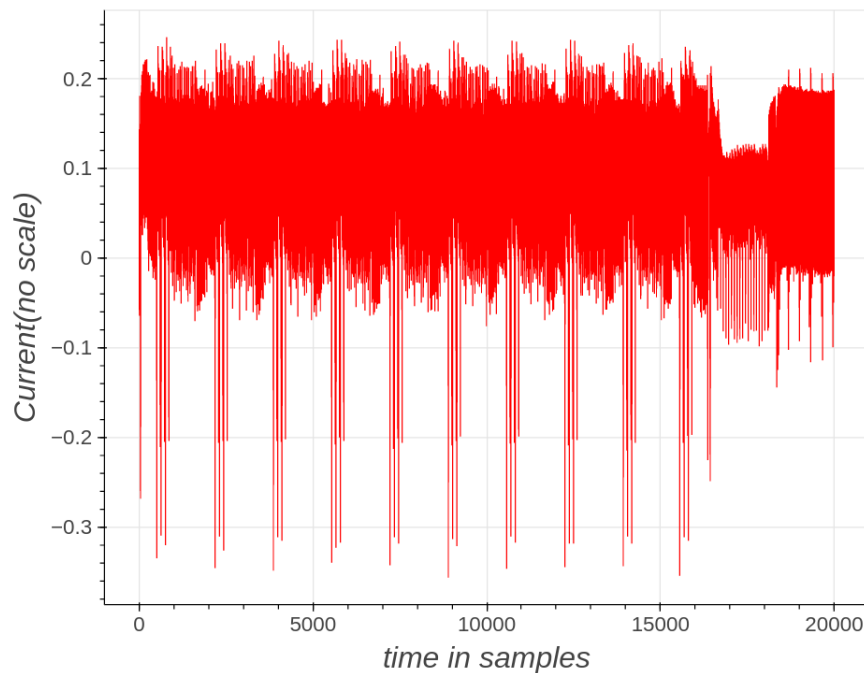


Figure 3.1: A collected power trace illustrating the 'signature' of 10 AES rounds.

So the DPA model is the one explicitly described in the previous chapter, where we need to generate the hypothetical leakage C' , split our power traces in sets of groups based on C' sin-

¹The ChipWhisperer target boards can be configured by the key of our choice through the SimpleSerial Python methods, but in our evaluation we do not 'cheat' by any means. We just use this information after the end of our analysis model to evaluate if our guess for the key was correct or not.

²The Y axis of the power trace is proportional to the AC-coupled consumption, also dependent on the gain settings of the Capture board amplifier and target board's shunt resistor.

³S-Box ShiftRows, MixColumns, AddRoundKey.

gle byte, calculate the difference of means across all groups of all sets and base our decision for the best key-guess on the average difference of means of all set(all steps exhaustively described in 2.1.1). The following function does exactly the latter, while taking in consideration the contribution of all 8-bits in power consumption, instead of single, in a single byte attack scenario (which resolves the ghost peaks problem 2.1.1).

```

1 def calculate_diffs_totaly(guess, byteindex=0):
2     grouped_byte_traces = []
3     for bit in range(8):
4         grouped_bit_traces = [], []
5         for trace_index in range(numtraces):
6             hypothetical_leakage = aes_internal(guess, textin_array[
7                 trace_index][byteindex])
8
9             #Mask off the requested bit
10            if hypothetical_leakage & (1<<bit):
11                grouped_bit_traces[0].append(trace_array[trace_index])
12            else:
13                grouped_bit_traces[1].append(trace_array[trace_index])
14
15            grouped_byte_traces.append(grouped_bit_traces)
16        diffs = []
17        for i in range(8):
18            means = np.average(grouped_byte_traces[i][0], axis=0), np.average
19                (grouped_byte_traces[i][1], axis=0)
20            diffs.append(abs(means[1] - means[0]))
21
22        diffs = np.average(diffs, axis=0)
23
24    return np.array(diffs)

```

Since we are attacking each byte of the key separately our attack needs to be repeated for 16 times, while our best guess for each sub-byte is the key guess with the largest spike in the already calculated absolute average difference of means of all sets. The following Python listing, does exactly that by calling the previous defined function, taking care of all Difference of Means calculations needed. As expected the attack obtained the full 16-bytes key correctly.

```

1 from tqdm import trange
2 import numpy as np

```

```

3
4 #Store your key_guess here, then compare to known_key
5 key_guess = []
6 known_key = [0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7,
7             0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c]
8
9
10 full_diffs_list = []
11
12 for subkey in trange(0, 16, desc="Attacking Subkey"):
13
14     max_individual_diffs = [0] * 256
15     full_diffs = [0] * 256
16     for guess in range(256):
17         diffs = calculate_diffs_totaly(guess,byteindex=subkey)
18         max_individual_diffs[guess] = np.max(diffs)
19         full_diffs[guess] = diffs
20
21     top_diffs = np.argsort(max_individual_diffs)[::-1]
22
23     key_guess.append(top_diffs[0])
24
25     full_diffs_list.append(full_diffs[:])
26     print('Subkey {:d} - most likely {:X} (actual {:X})'.format(subkey,
27         top_diffs[0], known_key[subkey]))
28
29
30 print("Top 5 guesses:")
31 for g in top_diffs[0:5]:
32     print('{:0X} - Diff = {:.f}'.format(g, max_individual_diffs[g]))

```

```

1 Attacking Subkey:   █6%|           | 1/16 [02:17<34:26, 137.78s/it]Subkey 0
   - most likely 2B (actual 2B)
2 Top 5 guesses:
3 2B - Diff = 0.011724
4 2A - Diff = 0.004716
5 22 - Diff = 0.003240
6 FB - Diff = 0.003207
7 40 - Diff = 0.003137
8 Attacking Subkey:  █12%|           | 2/16 [04:41<33:00, 141.45s/it]Subkey 1
   - most likely 7E (actual 7E)

```

```
9 Top 5 guesses:
10 7E - Diff = 0.012125
11 8E - Diff = 0.005620
12 AE - Diff = 0.005615
13 4C - Diff = 0.005426
14 68 - Diff = 0.005243
15 Attacking Subkey: ██████19%|          | 3/16 [07:00<30:25, 140.39s/it]Subkey 2
    - most likely 15 (actual 15)
16 Top 5 guesses:
17 15 - Diff = 0.008247
18 14 - Diff = 0.004608
19 A1 - Diff = 0.002908
20 5E - Diff = 0.002820
21 83 - Diff = 0.002739
22 Attacking Subkey: ████████25%|        | 4/16 [09:15<27:35, 137.98s/it]Subkey 3
    - most likely 16 (actual 16)
23 Top 5 guesses:
24 16 - Diff = 0.009201
25 17 - Diff = 0.005000
26 EC - Diff = 0.002889
27 11 - Diff = 0.002839
28 EA - Diff = 0.002832
29 Attacking Subkey: ██████████31%|      | 5/16 [11:33<25:18, 138.07s/it]Subkey 4
    - most likely 28 (actual 28)
30 Top 5 guesses:
31 28 - Diff = 0.009258
32 29 - Diff = 0.004705
33 43 - Diff = 0.003090
34 2 - Diff = 0.003054
35 42 - Diff = 0.003034
36 Attacking Subkey: ██████████38%|      | 6/16 [13:46<22:44, 136.48s/it]Subkey 5
    - most likely AE (actual AE)
37 Top 5 guesses:
38 AE - Diff = 0.009590
39 AF - Diff = 0.004485
40 4A - Diff = 0.002488
41 8E - Diff = 0.002397
42 5B - Diff = 0.002338
43 Attacking Subkey: ██████████44%|      | 7/16 [15:59<20:15, 135.09s/it]Subkey 6
```

```
    - most likely D2 (actual D2)
44 Top 5 guesses:
45 D2 - Diff = 0.011497
46 2 - Diff = 0.005522
47 14 - Diff = 0.005398
48 65 - Diff = 0.005376
49 C4 - Diff = 0.005073
50 Attacking Subkey: ██████████50%|      | 8/16 [18:06<17:42, 132.79s/it]Subkey 7
    - most likely A6 (actual A6)
51 Top 5 guesses:
52 A6 - Diff = 0.008035
53 A7 - Diff = 0.004575
54 1A - Diff = 0.002415
55 8F - Diff = 0.002399
56 93 - Diff = 0.002281
57 Attacking Subkey: ██████████56%|      | 9/16 [20:21<15:33, 133.37s/it]Subkey 8
    - most likely AB (actual AB)
58 Top 5 guesses:
59 AB - Diff = 0.005787
60 AA - Diff = 0.004893
61 B1 - Diff = 0.002271
62 E0 - Diff = 0.002236
63 7D - Diff = 0.002177
64 Attacking Subkey: ██████████62%|      | 10/16 [22:37<13:24, 134.03s/it]Subkey
    9 - most likely F7 (actual F7)
65 Top 5 guesses:
66 F7 - Diff = 0.009309
67 F6 - Diff = 0.004516
68 2 - Diff = 0.002518
69 A4 - Diff = 0.002246
70 71 - Diff = 0.002208
71 Attacking Subkey: ██████████69%|      | 11/16 [24:50<11:09, 133.81s/it]Subkey
    10 - most likely 15 (actual 15)
72 Top 5 guesses:
73 15 - Diff = 0.012953
74 14 - Diff = 0.004700
75 20 - Diff = 0.003531
76 F1 - Diff = 0.003413
77 8 - Diff = 0.003184
```

```
78 Attacking Subkey: ██████████ 75%| | 12/16 [27:09<09:01, 135.50s/it]Subkey
    11 - most likely 88 (actual 88)
79 Top 5 guesses:
80 88 - Diff = 0.011863
81 58 - Diff = 0.005388
82 4E - Diff = 0.005314
83 B3 - Diff = 0.005137
84 B2 - Diff = 0.005064
85 Attacking Subkey: ██████████ 81%| | 13/16 [29:19<06:41, 133.86s/it]Subkey
    12 - most likely 9 (actual 9)
86 Top 5 guesses:
87 9 - Diff = 0.005463
88 8 - Diff = 0.004577
89 9C - Diff = 0.003188
90 9D - Diff = 0.003082
91 4A - Diff = 0.002966
92 Attacking Subkey: ██████████ 88%| | 14/16 [31:27<04:24, 132.12s/it]Subkey
    13 - most likely CF (actual CF)
93 Top 5 guesses:
94 CF - Diff = 0.006228
95 CE - Diff = 0.004728
96 D6 - Diff = 0.002323
97 D5 - Diff = 0.002154
98 92 - Diff = 0.002095
99 Attacking Subkey: ██████████ 94%| | 15/16 [33:40<02:12, 132.21s/it]Subkey
    14 - most likely 4F (actual 4F)
100 Top 5 guesses:
101 4F - Diff = 0.008088
102 4E - Diff = 0.004635
103 B8 - Diff = 0.002548
104 DA - Diff = 0.002530
105 BA - Diff = 0.002448
106 Attacking Subkey: ██████████ 100%| | 16/16 [35:51<00:00, 134.48s/it]Subkey
    15 - most likely 3C (actual 3C)
107 Top 5 guesses:
108 3C - Diff = 0.009924
109 3D - Diff = 0.004569
110 FA - Diff = 0.002768
111 21 - Diff = 0.002761
```


112 60 - Diff = 0.002655

3.1.2 CPA against AES block cipher on XMEGA uC

It is obvious that the Analysis of the DPA needs a considerable amount of time for the key-recovery, as demonstrated in 3.1.1, especially for the refined approach proposed in Chapter 2. Although the conventional analysis of DPA reduced the time duration analysis down to 5 minutes, the problem of ghost peaks arises meaning that some sub-bytes of the key will be 'off' when compared to the actual key. For that reason CPA attacks are considered more powerful, as a typical analysis of this attack model usually requires up to 2 minutes, contrarily to the time consuming analysis of CPA. In 2.1.2 we proved the linear connection between the Hamming weight of C and the Power Consumption, while also discussed the reasoning behind finding the best sub-key guess (single byte attack) using the Pearson Correlation Coefficient.

So our attack is pretty straightforward; as before, the power consumption precedes the analysis phase. Note that this time 50 power traces¹, each representing a complete AES-128 ECB run, is enough for our attack to successfully obtain the encryption key, which is a significant improvement over the 5000 traces captured for the DPA 3.1.1. The reasoning behind 2.1.2 our best sub-key guess is concluded in the following Python listing, which is based on the correlation coefficient calculations for all possible key values of search space: 2⁸.

```

1 import numpy as np
2 from tqdm import trange
3 guess = [0] * 16
4 guess_corr = [0] * 16
5 trace_avg = np.average(trace_array, axis=0)
6 trace_std_dev = std_dev(trace_array, trace_avg)
7
8 for sub_byte in trange(0, 16, desc='Attacking full key'):
9     max_correlation = [0] * 256
10    for key_guess in range(0, 256):
11        hypothetical_hw = np.array([[HW[aes_internal(plaintext[sub_byte],
            key_guess)] for plaintext in textin_array]]).transpose()

```

¹We assume for each encryption individual run of total N runs to be fed with a random plaintext (input), while the encryption key of the AES is considered unknown and well secure.

```

12     hypothetical_hw_avg = mean(hypothetical_hw)
13     hypothetical_std_dev = std_dev(hypothetical_hw,
    hypothetical_hw_avg)
14     correlation = covar(X=trace_array, X_mean=trace_avg, Y=
    hypothetical_hw, Y_mean=hypothetical_hw_avg)
15     cpaoutput = correlation / (hypothetical_std_dev * trace_std_dev)
16     max_correlation[key_guess] = max(abs(cpaoutput))
17
18     guess[sub_byte] = np.argmax(max_correlation)
19     guess_corr[sub_byte] = max(max_correlation)
20
21 print('Key recovered as: ', bytearray(guess))
22 print('Correlations: ', guess_corr)

```

```

1 Attacking full key: ██████████100%|| 16/16 [00:11<00:00, 1.43it/s]Key
    recovered as: CWbytearray(b'ea 79 79 20 c8 71 44 7d 46 62 5f 51 85 c1
    3b cb')
2 Correlations: [0.876890415823039, 0.9261106049043423,
    0.8973921377237717, 0.8372741847284866, 0.9224813418582386,
    0.846444254334692, 0.8867622695106585, 0.7960763670512547,
    0.8909071967830633, 0.8634827804943137, 0.8523419213570385,
    0.7686746618067549, 0.878731192512089, 0.8977224254636939,
    0.8433309286719519, 0.7967759353977893]

```

Note that, ChipWhisperer software also provides Python classes and methods for a variety of leakage models f , facilitating all the CPA calculations¹, while also providing real time correlation calculations for the traces processed at the time². Figure 3.2 illustrates an 'automated' attack run against AES, utilizing ChipWhisperer Analyzer classes and methods calls on the generated 'attack' object. The table is refreshed every five power traces with new correlation values, over the current number of traces processed so far. However the ChipWhisperer Analyzer is not as optimized as LASCAR, which provides acceleration over conventional analysis calculations. In a latter sub-sections (3.2.1) we exploit LASCAR's analysis performance capabilities during the analysis phase of CPA attack against a hardware implementation of AES, as this will decrease the computation time, over thousands collected power traces, by

¹So there is no need for a user/evaluator to write his own code as we did.

²This is done by calling the `run(callback, update_interval)` method. The callback uses pandas DataFrames for visualization of the results, `update_interval` defines the number of traces to use (cumulative) before updating the correlation values of the sub-bytes.

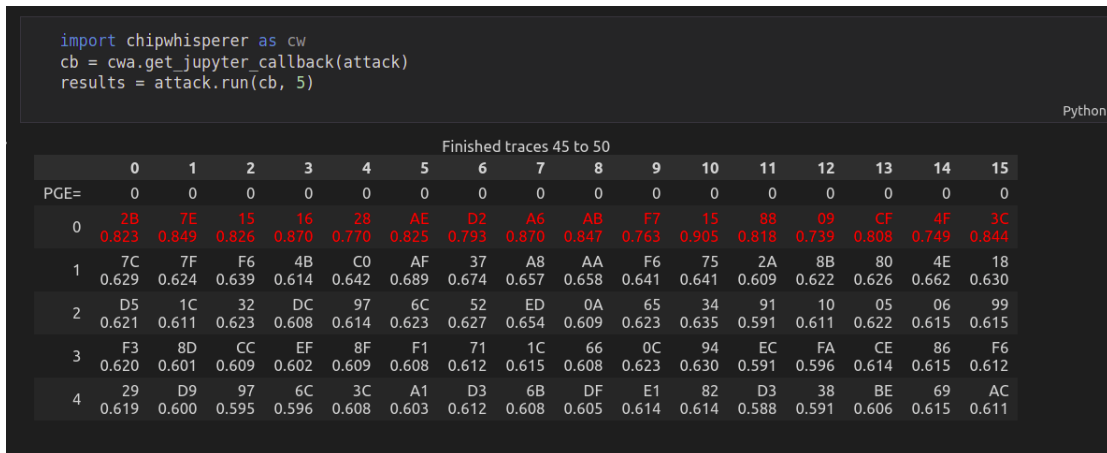


Figure 3.2: Using ChipWhisperer Analyzer for analysis phase, which utilizes Pandas DataFrames.

a factor of almost $\times 100^1$.

3.1.3 Bootloader AES-256

Bootloaders are simple programs that run on almost every microcontroller. Typically when a uC receives a command sequence, it is forced to enter the bootloader mode. So bootloader main objective is to 'authenticate' firmware updates/patches for an uC. This will ensure that an untrusted party cannot modify the contents of uC's flash memory or change its behaviour, causing protection/security issues. To prevent this from happening the already encrypted code is padded with a signature. The bootloader now has a way of authenticating the incoming firmware and discarded it if an inconsistency is found between the deciphered signature part/'domain' and bootloader's stored signature . So for a successful attack one needs to recover the encryption key, the Initialization Vector (IV) and the signature, in order to fake a 'trusted' firmware².

In this subsection we replicated the work of [25], where essentially we extended our CPA attack from AES-128 3.1.2 to AES-256, applied a DPA attack to recover the Initialization vector used in AES CBC mode of operation, utilized SPA techniques to recover the Signature used as authentication by the bootloader. Last but not least we proposed an effective countermeasure to prevent attackers from recovering the signature. By doing so the whole attack is compromised, since the bootloader will discard the 'fake' encrypted firmware sent

¹compared to the 'conventional' analyzer approach.

²Given that the bootloader uses AES CBC operation mode encryption.

by the attacker party since the authentication check will fail. The communication protocol to interact with the attacker is illustrated in Table 3.1. Note that a single 4-byte key is used instead of a hashed firmware¹, due to bootloader's constraints in size. Although the same 4-byte key data can be used to check the integrity of the transmitted data (data corruption occurred during transmissions 'errors' over the serial port), a separate 2 byte checksum is used in order to limit attack's surface, as an attacker cannot acknowledge that the firmware sent was accepted by the bootloader or not².

Header fixed (1-byte)	Signature (4-byte)	Encrypted Data (12 bytes)	CRC(2 bytes checksum)
-----------------------	--------------------	---------------------------	-----------------------

Table 3.1: Encrypted data format sent to the bootloader through a serial port with baud rate of 38400.

To prove the reproducibility of our results we created our own Jupyter Notebook, based on the seminal work's [25] notebook, which is also an open-source contribution to the ChipWhisperer's project. Essentially this notebook also works as a tutorial where the whole attack is explained thoroughly, while proposing a very effective countermeasure against key recovery, as discussed above, is the main task of the present sub-section.

From this point on we shift our attention to the signature recovery method and how an evaluator can prevent the realization of this attack. To comprehend the proposed countermeasure, one should be familiar with the SPA attack against the signature. For this reason we first demonstrate and analyze the timing attack against the signature ourselves in the following pages. After the bootloader has decrypted the incoming encrypted firmware, the received signature is checked against the bootloader's pre-saved signature. In case of a match, the new data are accepted and the uC source code is updated, otherwise the data are discarded. So the following pseudocode reveals how the received signature authentication is implemented in the bootloader's source code.

```

1 if (received_signature[0] == KNOWN_SIGNATURE_BYTE1) &&
2 (received_signature[1] == KNOWN_SIGNATURE_BYTE2) &&
3 (received_signature[2] == KNOWN_SIGNATURE_BYTE3) &&
4 (received_signature[3] == KNOWN_SIGNATURE_BYTE4) {

```

¹This is also not a public-key encryption where one can combine his private key with the hashed encrypted data.

²The bootloader only responds with two byte codes indicating data's integrity/validity.

```
5 accept the received firmware file
6 }
```

As this boolean expression will be short-circuited in C, this introduces a time dependency of how each of the 4-byte signature is checked. The time dependency is anticipated to be also propagated on the power consumption of the device, since as soon as a miss-match occurs on a single received signature byte all nested evaluations will not be executed affecting the power consumed by the bootloader. Essentially this is a SPA timing attack through power analysis. If the attacker has the capability to monitor the power consumption of the bootloader while the signature check is happening he/she should be able to visualize this time dependency mentioned on the power trace. Specifically 256 power traces are needed to be captured to recover the first byte of the signature, each trace is associated with a possible byte guess. The power trace representing the correct signature byte-guess is expected to be different from all other guesses. One way to make the latter property distinguishable is by calculating the difference between the average of all power traces and a power trace associated with a single byte guess. In Fig3.3 the latter statement is readily validated.

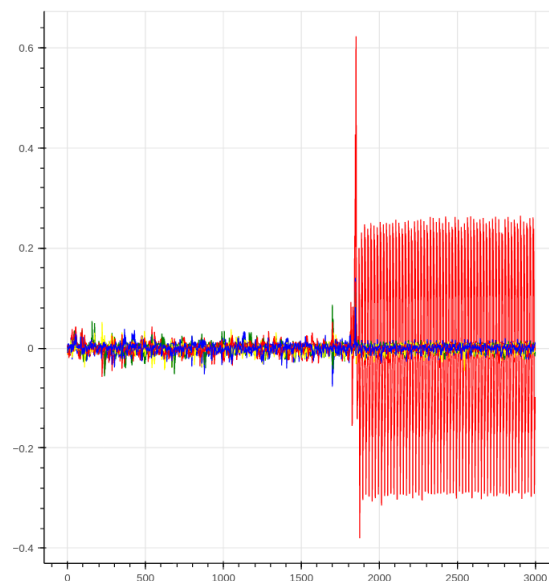


Figure 3.3: Difference between the average of all power traces and a single different power trace at a time, each illustrated with a different color to represent a different possible signature-value. The large red peak represents the difference between the mean of all power traces and the correct key guess of the first signature byte. The peak occurs at the point in time where the signature check takes place.

If the above approach is repeated for the remaining 3 signature bytes, the complete signature

can be obtained in 3 seconds. Our notebook also introduces a different metric for determining our best guess for a signature byte. The power trace associated with the correct signature guess is anticipated to have the lowest correlation to the mean of all traces.

As the attack described above is solely based on the time dependent signature checks, forcing the signature byte-checks to become time independent will result in attack's total failure. The following listing demonstrates our proposed solution for forcing signature-check evaluations to be time-independent.

```
1 store_sig0 = ((tmp32[0] == SIGNATURE1) ? 1:0);
2 store_sig1 = ((tmp32[1] == SIGNATURE2) ? 1:0);
3 store_sig2 = ((tmp32[2] == SIGNATURE3) ? 1:0);
4 store_sig3 = ((tmp32[3] == SIGNATURE4) ? 1:0);
5 store_result = store_sig0 & store_sig1 & store_sig2 & store_sig3;
6 if(store_result){
7   accept_the_received_firmware_file();
8 }
```

Listing 3.1: Modification of bootloader's source code as our proposed countermeasure against SPA attack on signature

If the attacker party attempts the SPA attack as demonstrated above, there is no evidence indicating a power trace with a deviance, to be selected as our best guess for a given signature-byte as illustrated in Figure3.4. Similarly the correlation between mean of all power traces and any given power trace representing a possible signature byte-value, will be identical and close to 1.0. This is pretty normal as independently of the incoming signature the bootloader will evaluate all 4 signature bytes, without 'exiting' prematurely resulting in similar power consumption for all possible signature byte-values.

3.2 Power Analysis attacks against hardware-based implementation of cryptographic schemes

For all our experiments described in the relevant subsections we need to acquire some power traces monitoring the device power consumption during complete runs of the targeted implementation, which can be explicitly triggered in demand, for arbitrary inputs (plaintexts), as mentioned in Chapter2. As in this section we are examining attacks against cryptographic

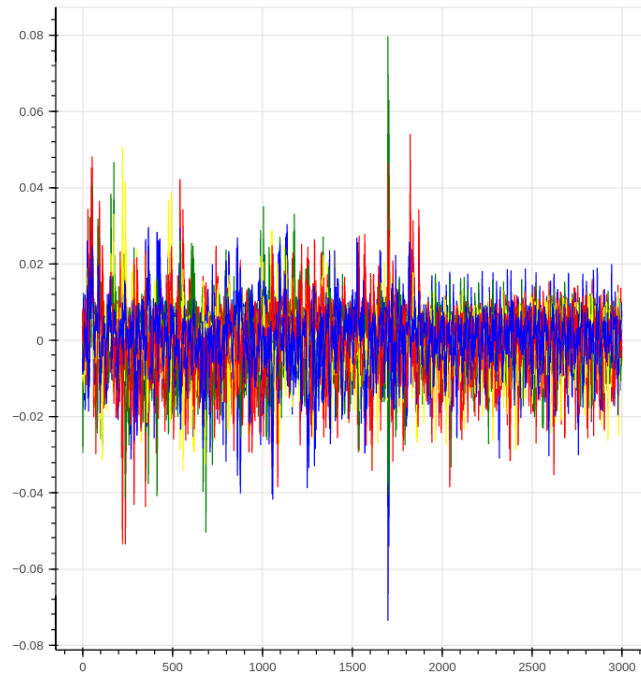


Figure 3.4: Difference between the average of all power traces and a single different power trace at a time, each illustrated with a different color to represent a different possible signature-value.

schemes implemented on Xilinx’s Artix FPGA, the board requires a bitstream file to be loaded in the device for its configuration. The bitstream files used in our experiments were generated using the Vivado toolchain and the RTL-description files (Verilog) provided from various sources for the crypto-core¹, while the bitstreams can also be obtained from ChipWhisperer relevant Github Repo. A great guide for anyone interested for a deep-dive in interacting with the Artix FPGA, often referred as CW305 in ChipWhisperer’s targets environment, is CW305_Hardware_Documentation.

3.2.1 CPA against AES block cipher on Artix7

In Chapter2 we have underlined the importance of selecting a suitable leakage model f for providing enough information through the power side-channel for a successful attack. For the CPA attack against a software implementation of AES, we proved that the first S-Box transformation is the ideal leakage model for mounting the attack. The reason for selecting the S-Box transformation in the first place, besides its non-linear properties, was that ” the power consumption of a typical uC is directly related to the intermediate values that

¹AES: AES_Google_Vault, ECC: CryptTech_ecdsa256.

are required to be loaded from some type of memory into a data bus with a certain capacitive behavior”, meaning that all the side-channel information needed will be embedded into the power traces associated with the S-Box transformation, while the search space for this demonstrated single-byte attack is computationally feasible(2^8). On the other hand hardware implementation of AES usually are much faster than software implementation having direct implication on our leakage model of choice for a successful attack. In particular the crypto-target/hardware-accelerator implementation of AES_Google_Vault meant to be mapped in our FPGA resources, is able to execute a complete AES_128 round in single cycle, contrarily to any given software-implementation requiring in the best case hundreds of clock cycles for a single AES round. Although, this significantly lower latency is generally preferable, from the attackers perspective this new model has dire consequences for the conventional leakage model of S-Box described in Chapter2 and utilized in 3.1.2. Figure3.5 is a high-level description of the our implementation under attack, where a AES round to complete takes exactly one clock cycle.

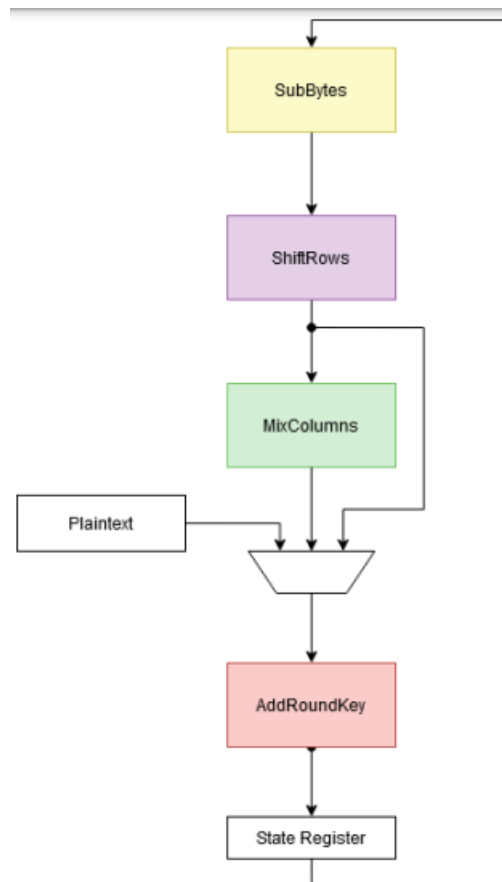


Figure 3.5: High-Level illustration of the targeted hardware AES implementation(figure’s source: CW305 Whitepaper)

You may probably already wondered if the Hamming Weight of S-Box is still capable of providing us with the side-channel leakage needed for a successful CPA attack. Well the answer is that this time the linear relationship between power and first round S-Box's Hamming Weight illustrated is heavily diminished as the power consumption of a single cycle, namely a single round is mainly affected by the storing operation of an intermediate round result to the state register. So for an effective CPA attack we need to shift our focus to the state register of Figure 3.5. Also note that in order for power to be consumed, the state register needs to be refreshed with a different value compared to the one currently latched/stored. For instance, in case of a register holding some arbitrary single-byte value linked with the state-result after the execution of 5th AES (let that be 0xCE), power will be consumed only if the state result of the next round is not equal to 0xCE. This consideration is interpreted in our model by selecting the Hamming distance between the previous and new state register value. The only thing left to find which two AES round states should be selected for our leakage model. Recall that in the last round of AES the MixColumns operation/transformation is absent. This is exactly what we need, namely the hamming distance between the output¹ and the input² of the last round. The reason behind the fact that the Hamming Distance between the first round input and output is not ideal, is due to the existence of MixColumns operation between the register transition. This operations brings diffusion of multiple byte of the intermediate AES state together³, resulting in an infeasible search space of 2^{32} , as now the attacker has to evaluate four bytes of the key simultaneously. To make things even worse this 'search' has to be repeated four times, one for each AES state-column.

The CPA attack against the AES running on the FPGA is actually no different from the one demonstrated in 3.1.2, except that we altered our leakage model of choice. However we need to capture up to 5000 power traces to be sure that our attack will be successful, as the power spikes of the Last Round's Hamming Distance tend to be quite small⁴.

After acquiring 5000 traces let's move on to the analysis phase. This time around we are going to use LASCAR, taking advantage of this repository optimized correlation coefficient

¹ciphertext

² $SBox^{-1}(ShiftRows^{-1}(ciphertext \oplus 10_th_Round_Key))$.

³Mixing 4 bytes of the same state column together

⁴Capturing that much traces also help with the noise.

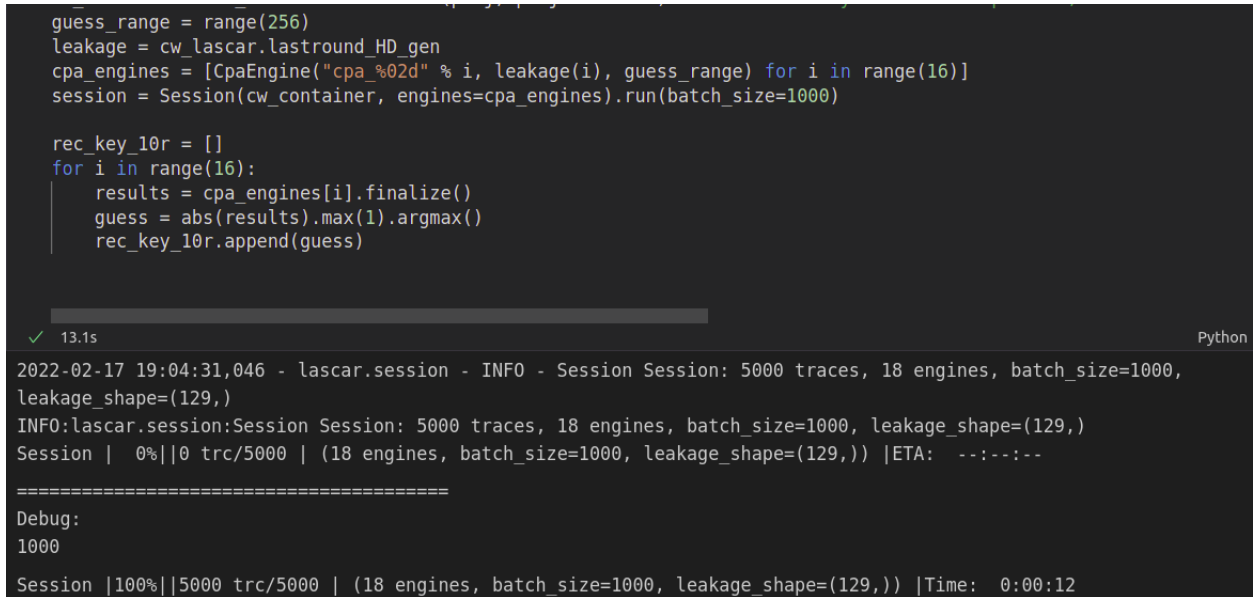
calculations the 10th-round key can be recovered in 12 seconds as seen in Figure 3.6. LASCAR/ChipWhisperer integration already provides the appropriate python method to select the Hamming Distance between the input ($SBox^{-1}(ShiftRows^{-1}(ciphertext \oplus 10_th_Round_Key))$) and output ($ciphertext$) of the last round as exhibited in the following listing.

```

guess_range = range(256)
leakage = cw_lascar.lastround_HD_gen
cpa_engines = [CpaEngine("cpa_%02d" % i, leakage(i), guess_range) for i in range(16)]
session = Session(cw_container, engines=cpa_engines).run(batch_size=1000)

rec_key_10r = []
for i in range(16):
    results = cpa_engines[i].finalize()
    guess = abs(results).max(1).argmax()
    rec_key_10r.append(guess)

```



```

✓ 13.1s Python
2022-02-17 19:04:31,046 - lascar.session - INFO - Session Session: 5000 traces, 18 engines, batch_size=1000,
leakage_shape=(129,)
INFO:lascar.session:Session Session: 5000 traces, 18 engines, batch_size=1000, leakage_shape=(129,)
Session | 0%|10 trc/5000 | (18 engines, batch_size=1000, leakage_shape=(129,)) |ETA: --:--:--
=====
Debug:
1000
Session |100%|5000 trc/5000 | (18 engines, batch_size=1000, leakage_shape=(129,)) |Time: 0:00:12

```

Figure 3.6: Utilize LASCAR python module to diminish CPA calculation time.

```

1 def lastround_HD_gen(byte):
2     def selection_with_guess(value, guess):
3         INVSHIFT_undo = [0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1,
4             6, 11]
5         st10 = value[INVSHIFT_undo[byte]]
6         st9 = inv_sbox[value[byte] ^ guess]
7         return hamming(st9 ^ st10)
8     return selection_with_guess

```

Listing 3.2: Source-code origin:ChipWhisperer Repo

The Figure 3.7 visualizes the 'final' correlation coefficients after processing all 5000 power traces. The colored 'waves' are associated with our best key guess correlation at each point in time. There are 16 distinct colors each representing the correlation for a single byte, while the peaks all occur concurrently at the point in time where the last round is executed, which is anticipated given our selected leakage model (Hamming Distance of last round). The non-colored waves represent the maximum correlation, both positive and negative, 'achieved' at the given point in time/sample.

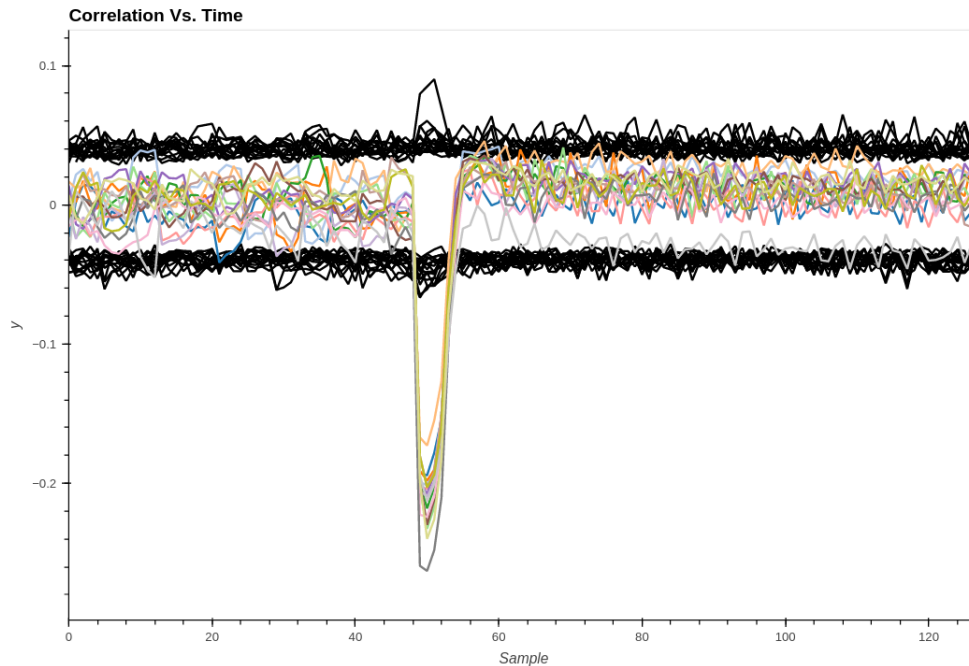


Figure 3.7: Plotting the best correlation achieved for all single key-byte guesses at a given time/sample.

An interesting task, from the attacker's perspective, is to figure out the least amount of traces needed to be acquired, that would still lead to successful key recovery without any problem. For that reason PGE (Partial Guess Entropy) is used. PGE is a way of defining how far is our best key guess from the actual key. A PGE of one for a given byte-guess means that there is only one sub-key guess¹ with a higher correlation than the actual sub-key. As a result for a successful attack we need for all single byte guesses a PGE of zero. In Figure3.8 we plot all sixteens PGEs associated with our best guess for the key, while increasing the number of power traces used in our calculations. From this plot we can easily deduce that a thousand of power traces will always satisfy the requirements for a victorious attack. However the PGE cannot be calculated if the attacker does not have knowledge of the actual key from the beginning of the attacks. So how can we still find out the minimum number of traces need for the attack to succeed?

The figure3.9 depict the deviation between the best key guess and the second best key guess, considering the number power traces used for calculating this deviation in the first place. By observing the graph we can assuredly pose that a thousand traces are enough, which also validates our observations made using the PGE metric.

¹Based on our correlation calculation over a certain number of power traces.

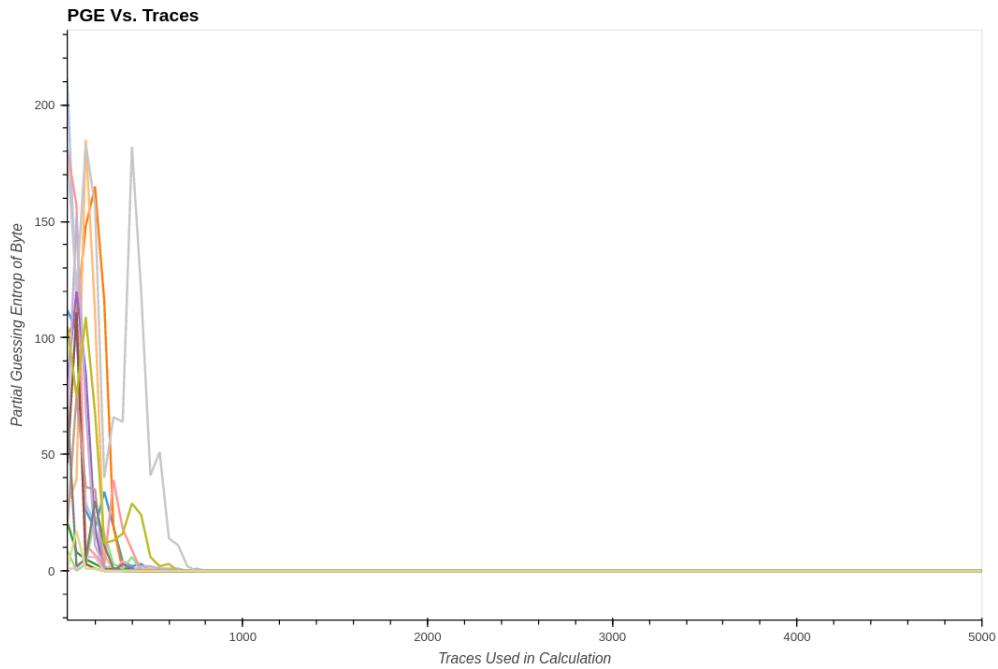


Figure 3.8: PGE over the number of power traces used so far.

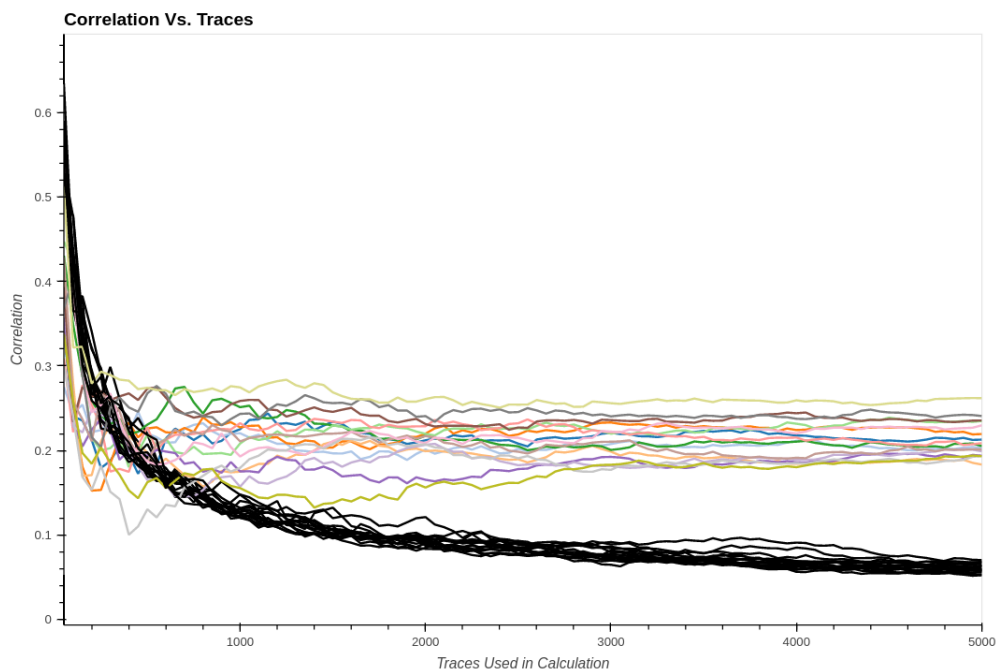


Figure 3.9: Colored waves represent the best key guess, black waves illustrate the second best key guess.

Before moving on, we should mention that hardware implementations of AES may vary significantly from each other, as a result different leakage model may arise as the most suitable

according to the given implementation. For instance, in [26, 27] a different approach is followed, where some bytes of the plaintext are held constant while the attack is launched during MixColumns first operation¹. For a hand-on attack curious readers are referred to Chapter 9 of CW305 Whitepaper also accompanied with a hands-on tutorial explicitly demonstrating the feasibility of this attack.

3.2.2 Whitebox Attack against Elliptic Curve Cryptography

In this experiment we replicated a Power Analysis attack presented in [28], against ECDSA public-key based, signature generation algorithm. The target board for this attack is Artix, ChipWhisperer's 'FPGA-based', while the capture board used, once again, is ChipWhisperer-Lite capture board (see Appendix). The reproducibility of the results in the aforementioned work was made possible thanks to the apparent intention of the authors to make their study disclosed, while also serving a seminal role for interested feature academic endeavors.

Dissimilarly to other cryptographic implementations, the one evaluated in this subsection leaks only a relative small exploitable information through the power side-channel. On top of that, this information is not spread throughout the full width of the collected power traces, meaning that it is concentrated in certain power samples in time. This is vividly demonstrated in the *Jupyter notebooks* used during this proof-of concept evaluation. Particularly Test Vector Leakage Assessment (TVLA) [29, 30] criterion, the most common metric used on applications/implementations for their side-channel resistance/ evaluation, was employed to confirm the latter. Note since the cryptographic operations related to the signature generation requires a consistent number of clock cycles to complete, while also each bit of the secret key (decisive for the signature generation) is processed in a fixed amount of cycles as well, timing attacks² are doomed. However the same principle allows the attacker party to obtain knowledge relevant to precise 'occurrence' of each single scalar key-bit processing. In this attack such knowledge was obtained successfully, through post PnR simulation of the dedicated public-key cryptographic core, eventually allowing us to recover ECDSA key used for the signature generation, by utilizing the obtained intelligence on the power analysis applied to the acquired traces respectively. Since our attack is based on obtaining critical details about the cryptographic operations timing relevant to scalar key-manipulations, such attack

¹1st AES Round MixColumns transformation.

²Through power traces analysis.

is considered as a *Whitebox* approach.

Table 3.2 illustrates the power traces required for a successful key recovery *campaign*, by comparing the number of incorrect single bit-guesses generated when applying the DoM & Correlation¹ power analysis approaches respectively. It should be noted that, this experiment does not reflect a real-world problem where the signature is refreshed after every signature-generation run; as in our evaluation the scalar-key value was kept fixed for the acquisition of all power traces, each representing a full signature-generation run.

Meaning that a real-attack can not average over several traces acquired to recover the key used for the signature generation. However by repetitively performing a *single trace attack* we figured out that on average only 45 scalar key-bits out of 256-bits were incorrect². Although this may not seem to be an encouraging sign for this *stand-alone* attack at first glance, the Hidden Number Problem³ allows us to recover the scalar-keys used during each signature generation runs. As described in [31] it is possible to recover the full 256-bit key used during ECDSA-P256 signature generation if in our *single trace attack* only a few successive/consecutive bits were guessed correctly. For the correct successive guessed-bits, if the criteria in [31] stand true, all 256-bits can be obtained within feasible computations and in a reasonable amount of time.

¹Comprehensive implementation details about the DoM & correlation analysis method can be found on the accompanied Jupyter Notebook.

²Using the more 'efficient' DoM approach during the power analysis phase.

³Solving the Hidden Number Problem is not a trivial task but also not infeasible.

Traces	DoM	Correlation
20	0	1
19	0	2
18	0	3
17	0	3
16	0	3
15	0	2
14	0	3
13	0	6
12	0	5
11	0	6
10	0	6
9	0	6
8	0	8
7	1	11
6	1	15
5	2	14
4	6	38
3	11	118
2	20	118
1	30	118

Table 3.2: Incorrect guesses for a given distinguisher as the power traces acquisition is decreasing

3.3 Countermeasures

Before closing this chapter once and for all we are going to discuss a few countermeasures that can be applied in any of the above attacks examined. The countermeasure we proposed in 3.1.3 is implementation specific, but recall the effectiveness of making all signature checks evaluation not to exit prematurely. A more generic approach/suggestions for undermining the effectiveness of the proposed attacks against AES is the following. Inserting a random delay in the intermediate AES transformation/operation, often referred as jitter, will most

probably lead in attack's failure. This is due to the fact that our attack is based on the premise that the information leaked from the side channel occurs at identical samples¹ in all power traces acquired. However this jitter can be bypassed by using 'synchronization techniques' which were also utilized, although not discussed in this thesis², in the CPA attack against the bootloader.

Other approaches [32, 33, 34] propose alterations to a typical VLSI design flow, in order to eliminate the correlation between the power consumption for a given operation and the processed data values. However such implementation typically require increased area & power sources. AES masking have shown promising results, providing 'enhanced' protection against CPA & DPA attacks, throughout the years [35, 36, 37]. Only recently there have been works employing Neural Network [38] and Machine Learning [39] approaches for attacking secure masked AES implementations, aligned with the popularity of the latter disciplines over the past few years. On the other hand quantum computing may pose a prospective threat against public-key cryptography as elucidated in [40, 41].

¹Points in time.

²Refer to our tutorial if interested in learning more on synchronization techniques.

Chapter 4

Fault Injection Attacks

Before moving forward with the actual attacks, since we have already described in detail the fundamental principles behind Differential Fault Analysis (Chapter2), which will be our attack model for our experiments, we are going to address the matter of clock glitch generation for our targets as described in Chapter2. We have already described the implication of glitches both in voltage supply and in the clock signal of a device. In Figure4.1 the glitching architecture of ChipWhisperer-Lite's Capture board is delineated.

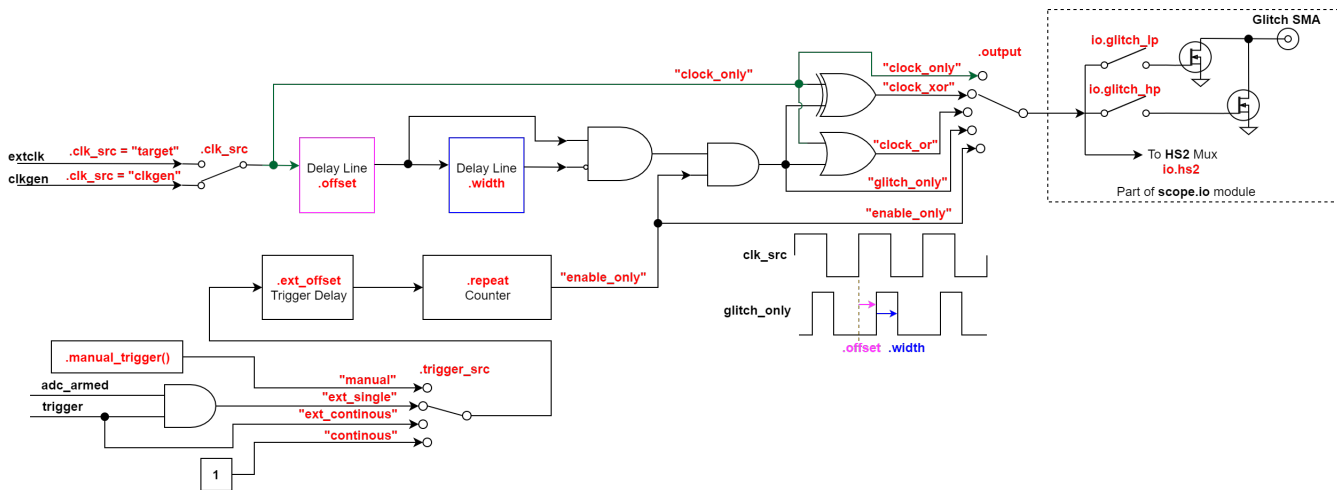


Figure 4.1: Block diagram of glitching 'sub-module' of Capture Lite Board (picture's source: ChipWhisperer ReadtheDocs)

In [23] the creator of ChipWhisperer Capture-Lite board demystifies the design of the clock glitching generation circuitry, which is based on using effectively the Digital Clock Manager of the capture's board FPGA in combination with Partial Reconfiguration techniques. So for the attacks examined the capture board feeds the device target under attack with a

glitchy clock. Three parameters are important for delivering a clock glitch precisely in the desired point in time¹. These parameters have to be configured appropriately, depending the device/implementation combination under attack, which are the following²:

- glitch width
- glitch offset
- glitch external offset.

Contrarily to the Power Analysis attacks examined in Chapter3, there are minor differences between attacks against hardware & software implementations. This is due to the fact that injecting two faults in each column of the AES state just before³ the last MixColumns operation, will always provide the faulty ciphertexts needed to apply the Differential Fault Analysis afterwards. This simple principle remain the same no matter if attacking a software or hardware AES implementation. For that reason we are not going to examine the attack against AES 'mounted' on our FPGA target separately from the attack on XMEGA microcontroller, since we only need to 'fine-tune' the three itemized important clock-glitching parameters above for our target/implementation combination judiciously. Finding the 'sweet-spot' parameters for injecting precisely the faults needed, for the last MixColumns attack-model, is a little more complicated and difficult when launching an attack against FPGA/AES combination. This is due to the fact that a single AES round has a duration of only one clock-cycle, thus the clock-glitching parameters have to be handle delicately.

4.1 DFA attack using phoenixAES/Clock glitching settings

So there are two steps in our attack:

1. Find the most suitable clock-glitch parameters.
2. Use the faulty ciphertexts for the Differential Analysis Phase.

¹For instance as seen in Chapter2 a successful fault-injection attempt causes a byte 'corruption' in the intermediate AES state anywhere between the last two MixColumns AES operation.

²See figure4.1 to understand the notion of width and offset, while every parameter is exhaustively documented in glitch documentation.

³Actually the fault can occur anywhere in-between the penultimate/8th Round MixColumns and last MixColumns operations.

The first step is actually trivial since we just need to 'search' in the 'sea' of feasible clock-glitch settings in order to inject faults just before the last MixColumns operation. In order to not sound repetitive let's jump directly to the attack against XMEGA uC. Before 'handing out' the parameter used, let's visualize what we are trying to achieve. In Figure?? the power consumption for a complete AES-128 encryption is captured. This figure is very indicative

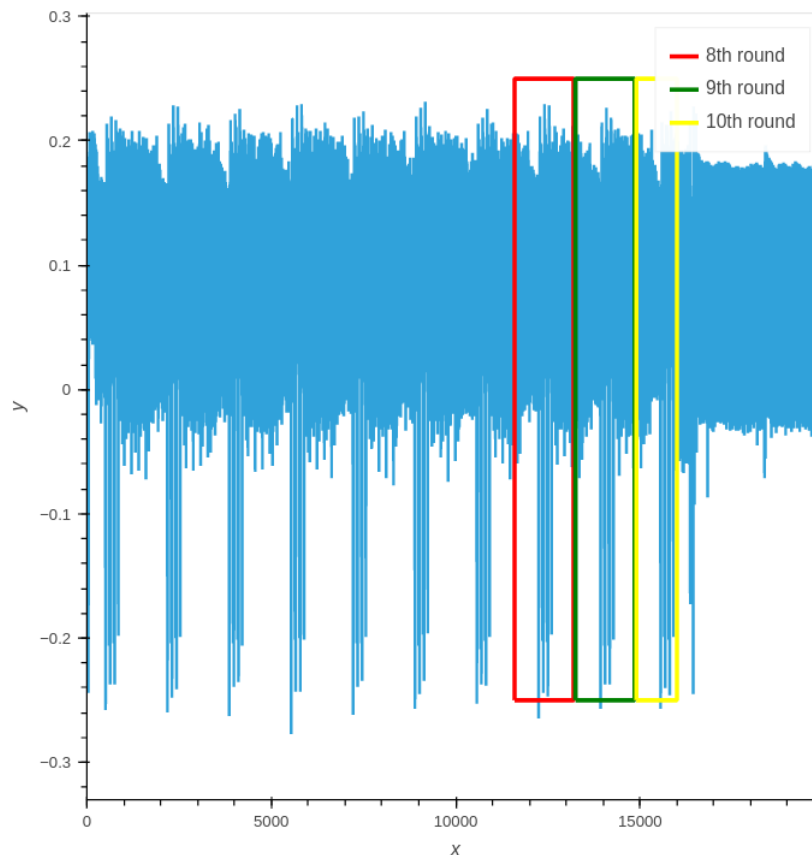


Figure 4.2: Capture all ten rounds of AES to visualize the point where fault need to be injected.

for which values can work for the external offset glitch parameter. A external offset off zero will result the glitch to be inserted right in the 0th sample, given that the 'power capture module' work with the same clock frequency. A external offset with a value of 13000 will delay the glitch by 13000 samples from the original trigger point, see Figure4.3. So a good starting point for our attack is sweeping the external offset value from 13000 to 14000. Width and offset parameters¹ tende to require different settings for a given uC/AES combination. So we can sweep between all possible value once, which is going to take some time, but then we can get a feedback about which value actually injected the required fault.

¹The value given to width and offset represent the shape of the glitch pulse to be generated in terms of percentage relative to one clock period.

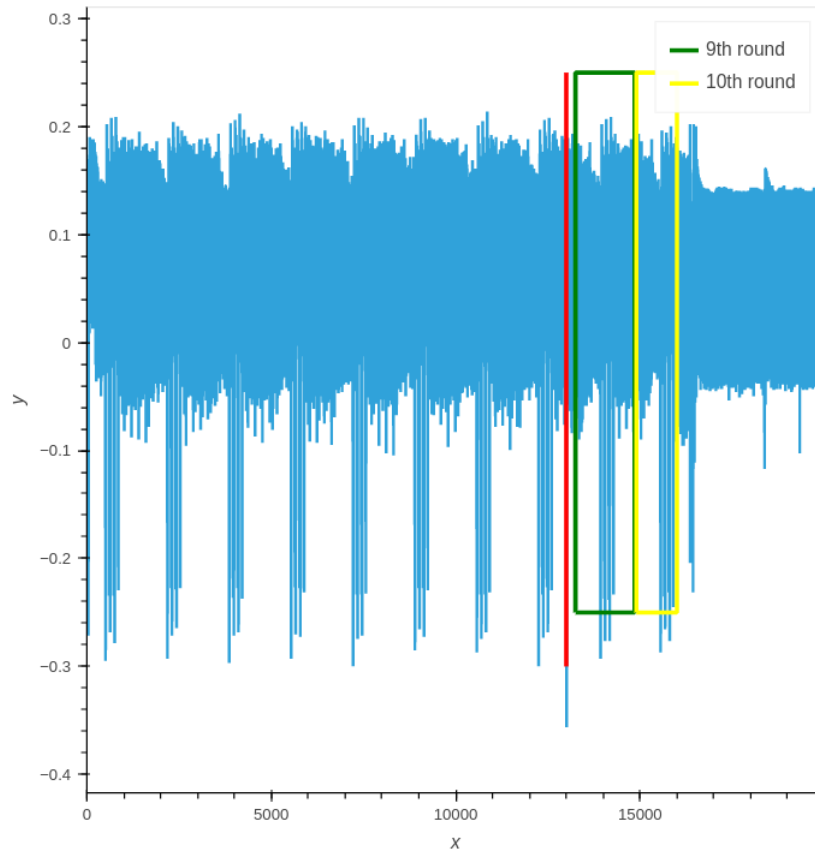


Figure 4.3: The red vertical line illustrates the 13000th sample around which the glitches are inserted.

For our experiments a glitch pulse of -6%, -5% width and -44%, -42% generated hundreds of faults for each column as illustrated in Figure 4.4. Classifying which column was influenced by looking at the faulty output generated is a trivial task¹. So step 1 everything good so far. Now we move to the more difficult part, the DFA phase. Here things get a little more complicated as proved when we inspected how Differential Analysis work in 2.2.2. No-one wants to deal with solving a non-linear system, so we mentioned how to discard incorrect solution with short-circuiting in code. Thanks to Philippe Teuwen for developing the phoenix-AES python script; a script dedicated for DFA whitebox implementation, that can recover AES last round's key in a matter of some thousands milliseconds.

¹There is a python method in our Github Repo, taking care of the ciphertext classification either as faulty or as 'golden'.

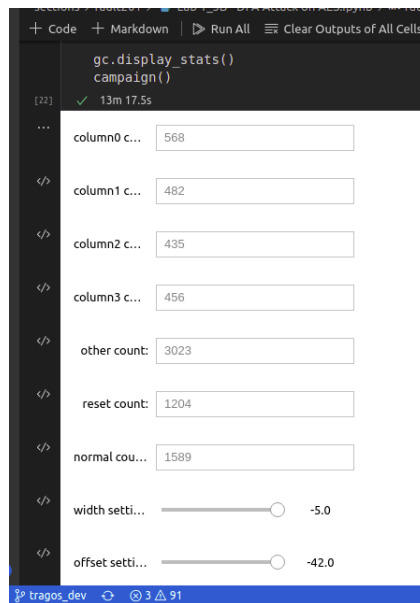


Figure 4.4: Glitch results of our 'campaign-attack' after sweeping through the glitch-parameters values provided in the text.

So the key recovery becomes trivial by calling the `crack_bytes` method (defined in phoenix-AES module) and giving the following parameters as input:

- `r9faults`: a list (or any other Sequence Type in Python) containing the faulty ciphertexts.
- `ref`: the 'golden'/correct ciphertext for a normal encryption run.

This method will return the correct AES last round-key, saving us from all the intense mathematical formalism. Again the some fundamental principals are applied in the glitch/DFA attack against the FPGA/AES attack.

Note that phoenixAES script also supports the attack model of 8th round MixColumn, referred as DFA R8. To make a long story 'short' suppose that a fault is injected just before the penultimate MixColumn operation (in 8th AES round). The faulty output generated by the fault in the encryption will result in a ciphertext with sixteen faulty bytes¹. In our Clock Glitch attack against FPGA/AES combination target we purposely injected faults before the 8th round MixColumns and used the `convert_r8faults_bytes` method ultimately leading in last round's key recovery².

¹compared to the 'golden' ciphertext.

²When passing the new faulty outputs to the `convert_r8faults_bytes` method, this call results in the well-known faulty outputs just as if the conventional 9th Round MixColumns attack-model was used.

4.2 Shortcomings of clock glitching / Countermeasures

The major drawback of the clock glitching attacks, as the one examined during the last section is the following. More complex devices use a Phase Locked Loop to acquire a new clean 'clock' signal instead of just using the external clock or if internal oscillators are selected as the clock source in the first place, the whole clock glitching approach is infeasible. So using the aforementioned components for obtaining a fault-free clock signal is effective against clock glitching attacks, for all device meeting the above clock 'recovery' requirements. For the clock glitching attack, we developed as part of this work, against the Artix FPGA discussed earlier, the glitchy clock signal fed to the device was provided by the Capture-Lite board. However the Artix board has its own internal oscillator and PLLs¹ so with the right configuration our target now has a way of building immunity to clock glitching. However ChipWhisperer capture board also has its own circuitry[42] that facilitates voltage glitching with the *crowbar* technique which introduces a ringing effect on device power line to generate faults[43]. To the best of our knowledge, during the time this thesis was written, there has not been yet a successful and consistent attack against Artix ChipWhisperer's 'FPGA-based' board using voltage glitches generated with the Capture-Lite board. So we have come close to achieving the latter, but still our voltage glitching parameters need to be searched in a 'sea' from successful & unsuccessful values, hopefully enabling us to be the first to demonstrate the attack and even open a pull request to contribute in ChipWhisperer Github tutorial repository. Defending against voltage glitching is a challenging task. The existence of few works, surveyed in [44], typical provide system-specific solutions. This may be an indication that voltage glitching may hunt us for a long time until a definite solution is given.

¹Input to the PLL can originate from the its crystal oscillator.

Chapter 5

Conclusion and Future Work

5.1 Conclusion & Contributions

In this thesis we proved how powerful Side-Channel attacks are, as they pose a serious threat for the security of cryptographic cores implemented on both software and hardware running on uC and FPGA respectively. All the attacks performed in this work underline the importance of security in embedded devices, while should motivate the industry to take the security for the devices built more seriously; especially since any individual can have access to the low-cost attack-equipment¹ as the one used in this Thesis (see Appendix). Hardware Security & Trust has non-ending capabilities for research as a newly established discipline.

Our study works as a reference for anyone interested in begging his journey to the world of Side-Channel attacks, as the fundamental principles and the theoretical background needed for Power Analysis & Fault Injection Attacks are concisely conveyed through Chapter2. In Chapters 3 & 4 we replicate real world attacks, disclosed in the relevant cited publications. Our main contributions to the existing work lies in the countermeasure proposed against the signature recovery thoroughly presented in 3.1.3; additionally a refined approach was proposed for DPA attacks, described in 2.1.1 and demonstrated in 3.1.1. Finally at the time this thesis was written, we have come close to achieving a functional and consistent voltage

¹The cost of buying all the equipment from NewAE Technology Inc./Colin 'O Flynn's startup. can be further abated by printing your own ChipWhisperer clone board, through an inexpensive PCB manufacturer, as the board layout can be derived from the 'original' schematic, which is publicly disclosed as part of the broader open-source project. Actually the schematics of all boards are available

glitching attack(see4.2).

5.2 Future Work

Nevertheless, most works in academia focus mainly on attacking the symmetric encryption. Our thesis barely touched the concept/topic of a power analysis attack against a public-key cryptography algorithm in 3.2.2, while the immense growth of deep learning can be aligned with the prospective 'research orientation' of Side Channel attacks. As deep learning approaches [45] only recently started to be used as an alternative and hopefully a more efficient approach in Power Analysis Attacks, eliminating the need for using any artificial/hypothetical approximation model during the analysis phase. A compelling extension of the present thesis, would include employing deep learning methods for side channel attacks against public-key cryptography. Conceivably any knowledge obtained by such attacks may then exploited for defending against quantum computing implications to public-key cryptography.

Bibliography

- [1] Swarup Bhunia and Mark Tehranipoor. Chapter 8 - side-channel attacks. In Swarup Bhunia and Mark Tehranipoor, editors, *Hardware Security*, pages 193–218. Morgan Kaufmann, 2019.
- [2] Frank Kagan Gürkaynak. *GALS system design: side channel attack secure cryptographic accelerators*. PhD thesis, ETH Zurich, 2006.
- [3] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.
- [4] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
- [5] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
- [6] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
- [7] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

- [8] Eyal Ronen, Colin O’Flynn, Adi Shamir, and Achi-Or Weingarten. IoT Goes Nuclear: Creating a ZigBee Chain Reaction. Cryptology ePrint Archive, Report 2016/1047, 2016. <https://ia.cr/2016/1047>.
- [9] YongBin Zhou and DengGuo Feng. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. Cryptology ePrint Archive, Report 2005/388, 2005. <https://ia.cr/2005/388>.
- [10] Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES ’02, page 2–12, Berlin, Heidelberg, 2002. Springer-Verlag.
- [11] Ross Anderson and Markus Kuhn. Tamper resistance: A cautionary note. In *Proceedings of the 2nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, WOEC’96, pages 1–11, USA, 1996. USENIX Association.
- [12] Colin O’Flynn. A framework for embedded hardware security analysis. 2017.
- [13] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.
- [14] Jing Pan, Jasper GJ Van Woudenberg, Jerry I den Hartog, and Marc F Witteman. Improving dpa by peak distribution analysis. In *International Workshop on Selected Areas in Cryptography*, pages 241–261. Springer, 2010.
- [15] Juncheng Chen, Jun-Sheng Ng, Nay Aung Kyaw, Ne Kyaw Zwa Lwin, Weng-Geng Ho, Kwen-Siong Chong, Zhiping Lin, Joseph Sylvester Chang, and Bah-Hwee Gwee. Normalized differential power analysis-for ghost peaks mitigation. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.
- [16] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

- [17] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *International conference on the theory and applications of cryptographic techniques*, pages 37–51. Springer, 1997.
- [18] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on AES. In *International Conference on Applied Cryptography and Network Security*, pages 293–306. Springer, 2003.
- [19] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference*, pages 513–525. Springer, 1997.
- [20] Jörn-Marc Schmidt and Christoph Herbst. A practical fault attack on square and multiply. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 53–58. IEEE, 2008.
- [21] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.
- [22] Micah Elizabeth Scott. Glitchy Descriptor Firmware Grab - scanlime:015. <https://www.youtube.com/watch?v=TeCQatNcF20>, <https://github.com/scanlime/facewhisperer>, 2016.
- [23] Colin O’Flynn and Zhizhang (David) Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. Cryptology ePrint Archive, Report 2014/204, 2014. <https://ia.cr/2014/204>.
- [24] Colin O’Flynn and Zhizhang (David) Chen. Synchronous Sampling and Clock Recovery of Internal Oscillators for Side Channel Analysis. Cryptology ePrint Archive, Report 2013/294, 2013. <https://ia.cr/2013/294>.
- [25] Colin O’Flynn and Zhizhang Chen. Side Channel Power Analysis of an AES-256 Bootloader. Cryptology ePrint Archive, Report 2014/899, 2014. <https://ia.cr/2014/899>.
- [26] Amir Moradi and Tobias Schneider. Improved side-channel analysis attacks on Xilinx bitstream encryption of 5, 6, and 7 series. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 71–87. Springer, 2016.

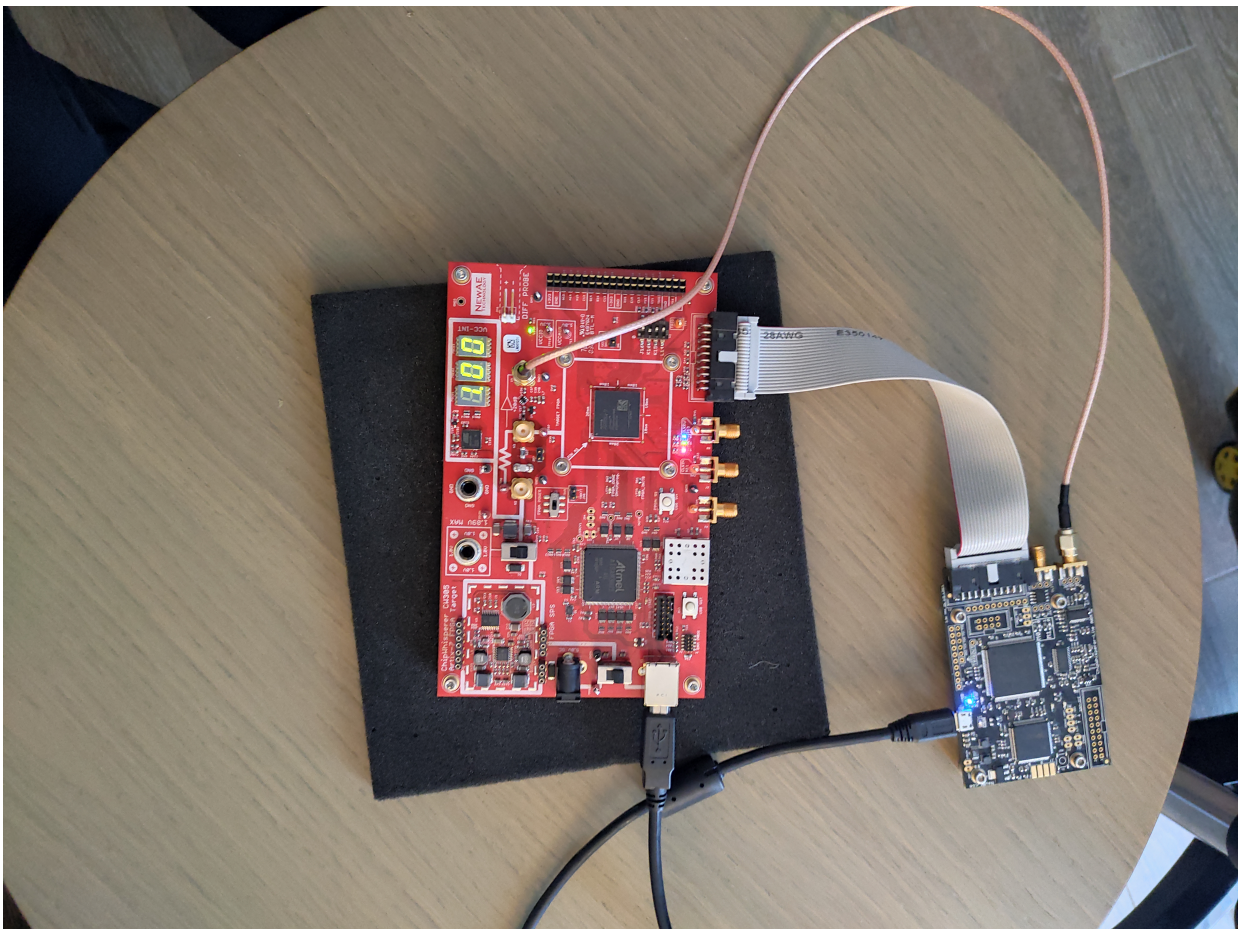
- [27] Aurelien Vasselle and Antoine Wurcker. Optimizations of Side-Channel Attack on AES MixColumns Using Chosen Input. *Cryptology ePrint Archive*, 2019.
- [28] Jean-Pierre Thibault, Colin O’Flynn, and Alex Dewar. Ark of the ECC: An open-source ECDSA power analysis attack on a FPGA based Curve P-256 implementation. *Cryptology ePrint Archive*, 2021.
- [29] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.
- [30] Michael Tunstall and Gilbert Goodwill. Applying TVLA to public key cryptographic algorithms. *Cryptology ePrint Archive*, 2016.
- [31] Victor Lomne and Thomas Roche. A Side Journey to Titan. *IACR Cryptol. ePrint Arch.*, 2021:28, 2021.
- [32] Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Proceedings of the 28th European solid-state circuits conference*, pages 403–406. IEEE, 2002.
- [33] Kris Tiri and Ingrid Verbauwhede. A VLSI design flow for secure side-channel attack resistant ICs. In *Design, Automation and Test in Europe*, pages 58–63. IEEE, 2005.
- [34] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.
- [35] Weize Yu and Selçuk Köse. A lightweight masked AES implementation for securing IoT against CPA attacks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(11):2934–2944, 2017.
- [36] Johannes Blömer, Jorge Guajardo, and Volker Krümmel. Provably secure masking of AES. In *International workshop on selected areas in cryptography*, pages 69–83. Springer, 2004.

- [37] ANSSI-FR. Masked AES implementations for Cortex M3/M4 microcontrollers. <https://github.com/ANSSI-FR/SecAESSTM32>, 2019.
- [38] Richard Gilmore, Neil Hanley, and Maire O’Neill. Neural network based attack on a masked implementation of AES. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 106–111. IEEE, 2015.
- [39] A machine learning approach against a masked AES, author=Lerman, Liran and Bon-tempi, Gianluca and Markowitch, Olivier. *Journal of Cryptographic Engineering*, 5(2):123–139, 2015.
- [40] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology ..., 2016.
- [41] Vasileios Mavroeidis, Kameer Vishi, Mateusz D Zych, and Audun Jøsang. The impact of quantum computing on present cryptography. *arXiv preprint arXiv:1804.00200*, 2018.
- [42] Colin O’Flynn. Fault injection using crowbars on embedded systems. *Cryptology ePrint Archive*, 2016.
- [43] Loic Zussa, Jean-Max Dutertre, Jessy Clediere, and Bruno Robisson. Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 130–135. IEEE, 2014.
- [44] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.
- [45] Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. X-DeepSCA: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

Appendix

ChipWhisperer Platform/Tools

1 Artix FPGA board w/ Capture Lite board



2 XMEGA microcontroller w/ Capture Lite board

