CRYPTOGRAPHIC IMPLEMENTATION RESISTANT TO SIDE CHANNEL ATTACKS ON RECONFIGURABLE HARDWARE

by
Mohak Bhatia

A dissertation submitted to the Department of Electrical and Computer Engineering,

Cullen College Of Engineering

in partial fulfillment of the requirements for the degree of

Masters Of Science
in Electrical and Computer Engineering

Chair of Committee: Dr. Hung "Harry" Le

Committee Member: Dr. Hien Nguyen (Co-Chair)

Committee Member: Dr. Yuhua Chen

University of Houston
May 2021

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere thanks to my committee chair Dr. Hung"Harry" Le for giving me the opportunity in such a challenging field. He taught me the methodology to carry out and present the research work as clearly as possible. It was a great privilege to do research work under him.

I cannot begin to express my thanks to Dr. Hien Nguyen for supporting me throughout my thesis by acting as the co-chair and for helping me write my dissertation.

I would also like to extend my deepest gratitude to Dr. Yuhua Chen for being my mentor and introducing me to the wonderful field of digital and hardware design. I am truly indebted to her for her teachings.

Special thanks to Colin O' Flynn & Alex Dewar for introducing to the exciting field of side channel analysis. Your hardware and software tools helped me immensely in understanding the nuances of this field.

ABSTRACT

The advent of the Internet of things has revolutionized the way we view the infrastructure of information technology and constantly pushes the boundaries of amalgamating the physical world with computer-based systems. For instance, allowing objects to sense and be controlled remotely across a network and forming technology hubs like smart grids, smart homes, virtual power plants, and smart cities.

However, in today's data-intensive computation-driven services, security and data privacy is perhaps the largest of the pitfalls. To give an example to support my assertion, according to Business Insider intelligence survey which was conducted in the last quarter of 2014, 39 percent of respondents felt that security is the biggest concern in adopting Internet of things technology. Security, trust, and privacy have always played a crucial part in computer security and with the advent of technologies like Internet of things & Cyber physical systems, recently the number of devices connected to the internet has gone up, thus the need for security has also increased.

The art of keeping messages secret is cryptography, while cryptanalysis is a study attempting to defeat cryptographic techniques. Strong cryptographic algorithms are just the beginning for securing your device. Current cryptographic algorithms have very high standards of security. For embedded devices connected to a network uses a special type of cryptographic algorithms called lightweight cryptography systems, which are highly resource efficient.

However, these cryptographic implementations are not as secure as full-fledged algorithms used in computer systems. There is something called as power profile which can leak information from the card. There are software and hardware implementations of the cipher which can lead to timing attacks. To tackle this We propose an AES cryptographic implementation that is resistant to power side channel attacks.

The deliverable from this thesis is the implementation of AES algorithm on a

32-bit Cortex M4 microcontroller and then applying three different side channel attack techniques: differential power attack, correlation, power attack and differential fault analysis techniques to leak the key of AES-128 so that security of the device is compromised and proposing two countermeasures to make the AES implementation resistant to side channel attacks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

The progress of computing and network technology is a harbinger of an era where our gadgets and devices are embedded with electronics, software, sensors, actuators, and connectivity that enable these objects to collect and exchange data at all times from anywhere across the globe. The ubiquitous presence of embedded devices connected to the network, which is the idea of IOT, provides an opportunity for malicious attackers to steal and manipulate information. Their security lags far behind those of PC computers. Unlike general purpose computers, the majority of IOT devices have very rudimentary UI, hence if an attack occurs, the user may be unaware of security of the device being compromised.

One way of resolving this is to use cryptographic modules for security in such devices. The Advanced Encryption Standard (AES) became the standard for encryption to protect sensitive information. With the increasing use of portable and wireless devices and demanding information security needs in embedded systems, we prompted efforts to find fast software-based implementations of AES encryption/decryption capable of running on resource-constrained environments in terms of processor speed, code space,energy usage and in particular those portable devices that have 32-bit ARM Cortex-M4 processor.

However, there are attacks known as side channel attacks which leak information during the normal functioning of the device and generally without any trace of the device being compromised. These attacks target the implementation of the cryptographic algorithms and not the cryptographic algorithms themselves, which are generally very strong and cannot be broken into. These attacks are noninvasive in nature, i.e., the attacks deal with power and timing information.

1.1 Motivation

It is nearly impossible to break AES using traditional methods such as brute force. Each subkey has 2^8 possible values which makes the number of possible key value and atleast 2^{128} total key combinations. However, using the side channels that are released while AES is running can make revealing the actual key quite convenient for the attackers.

The efficacy of the side channel attack brings an impetus to find countermeasures against it. We find the cryptographic algorithm being used in abundance nowadays in many ranging from RFID tags, smart cards, autonomous cars, secure enclave in our PCs. However, the security of these devices [4] [3] [16] are regularly compromised.

Even though all commercially available are functionally verified and tested for ensuring correct functionality, they are mostly not verified for security. Therefore, more research in designing secure chips and stronger ciphers are required.

In the past years, most of the studies about Side-Channel Analysis and Fault Analysis countermeasures have been carried out independently. The research direction of combined countermeasures - that is, countermeasures against both [2] [14] Side Channel Analysis and Fault Analysis is quite young and experimental. The difference between side-channel analysis and fault analysis has become much smaller than before, and the need for countermeasures that simultaneously address side channel leakage and fault sensitivity is becoming increasingly crucial.

1.2 Problem Statement

The research on countermeasures for SCA and FA is very rare and only recently has peaked the interest of researchers. This thesis aims to find countermeasures for differential power attacks and fault attacks on the AES-128 implementation on a 32-bit Cortex M processor. It targets specific countermeasures related to clock and

power randomisation. We first implement the cipher on the AES-128, after that we try to retrieve the 16 byte key using the leakage information that gets revealed using the power trace which is measured during repeated encryption operations on random plaintexts and a fixed key. The key is revealed by studying the peaks of the mean trace arrays. Later we determine the countermeasures for these attacks. We explore the different design trade-offs of these countermeasures like signal-to-noise ratio, number of power traces, and clock glitch width.

1.3 Thesis Roadmap

The thesis dissertation inspects concepts for the successful implementation of different side channel attacks and their corresponding countermeasures for preventing them.

Chapter 2 will introduce the related work that we referred to while working on my thesis. That includes research papers, journal papers, conference papers, and books.

In Chapter 3, we talk about the experimental setup and the methodology we used to implement different side channel attacks that we perform on 8-bit and 32-bit ARM target. we perform differential power attack, correlation power attack & differential fault attacks to reveal the AES key as a knowledgeable insider. We discuss the results and limitations of each of these attacks as well.

Chapter 4 talks about two different countermeasures that we implement to prevent attacks and create a hindrance for the attacker that will prevent him from breaking the AES implementation. We modify the unprotected AES to achieve this and then finally in chapter 5 we discuss the conclusions we derive from my thesis.

2 BACKGROUND & RELATED WORK

2.1 Cryptography Overview

The art of keeping messages secret is cryptography, while cryptanalysis is the study attempted to defeat cryptographic techniques. Cryptography is used to protect information from illegal access. It largely encompasses the art of building schemes (ciphers) which allow secret data exchange over insecure channels. The need of secured information exchange is as old as the civilization itself.

It is believed that the oldest use of cryptography was found in non-standard hieroglyphics carved into monuments from Egypt's Old Kingdom. In 5 B.C. the Spartans developed a cryptographic device, called scytale to send and receive secret messages. The code was the basis of transposition ciphers, in which the letters remained the same but the order is changed. This is still the basis for many modern day ciphers. The design of cryptographic protocols for securely proving one's identity has been an important aspect of modern cryptography. The primitive operation of cryptography is hence encryption. The inverse operation of obtaining the original message from the encrypted data is known as decryption. Encryption transforms messages into a representation that is meaningless for all parties other than the intended receiver. Almost all cryptosystems rely upon the difficulty of reversing the encryption transformation to provide security to communications. Cryptanalysis is the art and science of breaking the encrypted message. The branch of science encompassing both cryptography and cryptanalysis is cryptology and its practitioners are cryptologists. One of the greatest triumphs of cryptanalysis over cryptography was the breaking of a cipher. For many years, many fundamental developments in cryptology outpoured from military organizations around the world. One of the most influential cryptanalytic papers of the twentieth century was William F. Friedman's monograph entitled

The Index of Coincidence and its Applications in Cryptography. For the next fifty years, research in cryptography was predominantly done in a secret fashion, with a few exceptions like the revolutionary contribution of Claude Shannon's paper"The Communication Theory of Secrecy Systems" [15] , which appeared in the Bell System Technical Journal in 1949.

However, after the world wars, cryptography became a science of interest to the research community. The Code Breakers by David Kahn produced a remarkable history of cryptography. The significance of this classic text was that it raised the public awareness of cryptography. The subsequent development of communication and hence the need of privacy in message exchange also increased the impetus for research in this field. A large number of cryptographers from various fields of study began to contribute leading to the rebirth of this field. Horst Fiestel began the development of the US Data Encryption Standard (DES) and laid the foundation of a class of ciphers called as private or symmetric key algorithms. The structure of these ciphers became popular as the Fiestel Networks in general. Symmetric key algorithms use a single key to both encrypt and decrypt. To establish the key between the sender and the receiver, they are required to meet once to decide the key. This problem is commonly known as the key exchange problem, which was solved by Martin Hellman and Whitfield Diffie in 1976 [11]in their ground-breaking paper New Directions. Cryptology has evolved further with the growing importance of communications and the development of both processor speeds and hardware. Modern-day cryptographers have thus more work than merely jumbling up messages. They have to look into the application areas in which the cryptographic algorithms have to work. The transistor has become more powerful. The development of the VLSI technology have made the once cumbersome computers faster and smaller. The more powerful computers and

devices allow more complicated encryption algorithms to run faster. The same computing power is also available to cryptanalysts who will now try to break the ciphers with both straight forward brute force analysis, as well as by leveraging the growth in cryptanalysis. The world has thus changed since the DES was adopted as the standard cryptographic algorithm and DES was feeling its age. Large public literature on ciphers and the development of tools for cryptanalysis urges the importance of a new standard. The National Institute of Standards and Technology (NIST) organized a contest for the new Advanced Encryption Standard (AES) in 1997. The block cipher Rijndael emerged as the winner in October 2000 because of its features of security, elegance in implementation, and principled design approach. Simultaneously, Rijndael was evaluated by cryptanalysts and a lot of interesting works were reported. Cryptosystems are inherently computationally complex and in order to satisfy the high throughput requirements of many applications, they are often implemented by means of either VLSI devices or highly optimized software routines. In recent years, such cryptographic implementations have been attacked using a class of attacks which exploits the leaking of information through side channels like power, timing, intrusion of faults etc. In short, as technology progresses, new efficient encryption algorithms and their implementations will be invented, which in turn shall be cryptanalyzed in unconventional ways. Without doubt, cryptology promises to remain an interesting field of research both from theoretical and application points of view.

2.2 Symmetric Cryptography

The field of cryptography is divided into two groups based on the use of keys. In symmetric cryptography, we generally consider the same key for encryption and decryption, similar to how a classical door functions. The key is shared with the involved parties beforehand.

2.2.1 Galois Fields

In cryptography, we are interested in fields with a finite number of elements, which we call finite or Galois fields. A field is defined using the following theorem

Theorem: *A field with order m exists if m is a prime power i.e* $m = p^n$*, for some positive integer n and prime integer p. p is called the characteristic of the finite fields.* Roughly speaking, a Galois Field is a set with a finite number of elements. For example, GF(128), which can also be referred to as GF(2^8), has elements only 0 and 1 as p = 2.

Binary Finite Fields: Binary Fields is defined by set of two commutative operations(**.,+**) In which two elements can be subtracted, multiplied, divided by a nonzero element, multiplication distributes over multiplication. They are also refereed to as Galois Fields. The addition operation as used is XOR in BFF. *Modes of Operations:* The mode of operation describes how the block cipher of varying size can be repeatedly applied to the same key more than once We are provided with a nxn square matrix of the plaintext which goes to many rounds of transformations.

2.2.2 Cryptosystem

A cryptosytem is a set of five aspects: P,C, K, E, D

- P A set of finite plaintexts
- C A finite set of ciphertexts
- K Keyspace, or a finite set of keys.
- E Encryption Rule: $e_k \epsilon$E
- D Decryption rule: $d_k \epsilon$D

Each $e_k : \mathbf{P} \Rightarrow C, d_k : C \Rightarrow P$ are functions such that $d_k(e_k(x)) = x$

There are two broad categories of cryptosystems : Symmetric key and Asymmetric key. The encryption(public) and decryption(private) keys are the same in symmetric and different in asymmetric key. The private key is very hard to decipher in case of asymmetric cryptosystem. The number of rounds is determined by the cipher's resistant to known attacks.

2.2.3 Block Cipher

A block ciphers are used for bulk data encryption. It encrypts n bits of plain text data called as blocks. Plain text is divided into chunks of data and then applied block cipher. This Process can be repeated block by block or through the use of chaining. This encryption applies to a block of data and produces encrypted data of similar size. It uses a mapping from plain text to cipher text using a key. Decryption function uses the same process in reverse to get a block of pain text data from the encrypted data.

2.2.4 Structure of Block Ciphers

Internal of block ciphers can be seen below. The encryption process uses a key whitening step obtained using the key reversing process, for example, XOR function. After key whitening, there are a number of round keys which use key scheduling algorithms to obtain them. These roundings use a similar process as the key whitening process. For example, an integer modulo function can be used.

After n rounds, you get the cipher text. The designer hides the round keys , its process, key whitening process, and only shows the cipher text and plain text blocks to the public. If any intermediate data can be seen by an adversary, then the security cannot be guaranteed. Side channel attacks can reveal the intermediate data which will be discussed later.

The key and the block length can be independent of each other. Generally, it is 128, 192, 256 bits. The larger the key size, the more difficult it is to decrypt AES using the brute force method. An encryption round consists of three operations:

- Addition of round key
- *Confusion* The process of adding a S-Box or hiding the relation between the cipher text and the key.
- *Diffusion* Hiding the relation between the ciphertext and plain text.

2.3 Advanced Encryption Standard (AES

A worldwide standard since October 2000, There are different types of AES based on the key length & number of rounds.

Table 1: Types Of AES.

Type	Key Length	No. Of Rounds
AES-128	16 bytes	10
AES-192	24 bytes	12
AES-256	32 bytes	14

Each round has different steps of XORing the plain text with a round key which is typically done using *Round key addition* in Randomness layer. Confusion layer uses *byte substitution.* Final round which is diffusion layer uses *Shift row and Mix Column* technique.

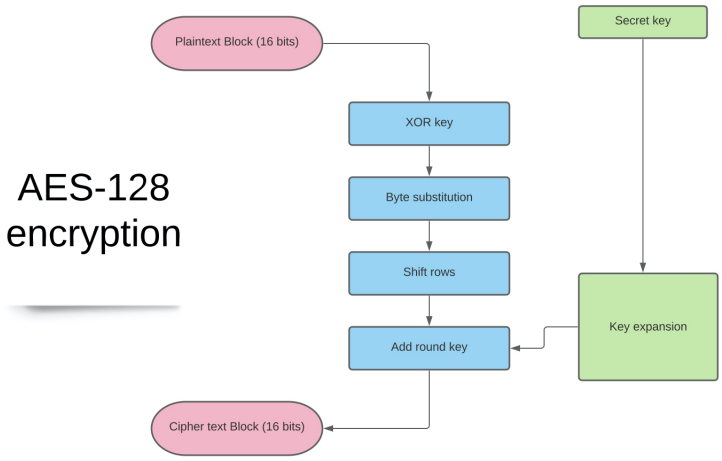

Figure 1: Structure of AES-128 Algorithm.

2.4 Physical Attacks

In this section, we talk about the technical background relevant to my work related to physical attacks. We will be providing descriptions for mainly two kinds of attacks, Side Channel Attack (SCA) and Fault Attacks (FA). We then elaborate and divide these attacks into Differential Power Attacks & Correlation Power Attacks (CPA) and Differential Fault Analysis (DFA). Later, we will be elaborating on the countermeasures for these attacks and testing their efficacy. Later, we test our design with something known as Test Vector Leakage Assessment (TVLA) for a thorough security assessment. The growth of cryptographic devices has been increasing since the advent of IOT. These range from smart cards, RFID tags, pacemakers, autonomous cars, home automation, and many others. The security of these devices is constantly a threat with the attacks rapidly increasing and becoming easier to produce. Conventional processors present in these devices are highly vulnerable to attacks. Even though these chips go through rigorous verification, which is critical for the correct functionality of the device, they do not necessarily go through such strict security evaluations. Hence, investigations into finding countermeasures are needed. The challenge in tackling side channel attacks comes from the fact that it violates the

classical notions of cryptography and are not equipped for hindering such attacks. Therefore, the danger of side channel attacks is more due to the fact that they can compromise and expose the fragility of such strong ciphers and make it unsuitable to secure the data of these applications. Thus, we need a thorough understanding of cryptographic ciphers.

2.4.1 Adversary Models

Three well-known and broad adversary models are the Black-Box model, the White-Box model, and the Grey-Box model. This classification is based on the amount of information that is available to the adversary. Three well-known and broad adversary models are Black-Box model, White-Box model, and Grey-Box model. This classification is based on the amount of information that is available to the adversary.

Black-Box Model

The first adversary model assumes that the attacker has access to the inputs and outputs of the cryptosystem only like how well they resist cryptanalytic or mathematical attacks. Good algorithms have emerged from the interplay between designers and attackers (i.e., cryptanalysts) and one such example is the Advanced Encryption Standard. Ideally, the only option an attacker has with such algorithms. Ideally, the only option an attacker has against such algorithms is an exhaustive search of all keys, known as a brute force attack. If the key space is large enough, this search will be impossible. While the symmetric-key encryption algorithms we consider are deemed unbreakable by modern standards, cryptanalysis is still an evolving research area, but out of the scope of this work.

White-Box Model

The second adversary model assumes that the attacker has access to all information of the implementation. Protecting against such an adversary requires an

obfuscated implementation. Naively, one could achieve this by hardcoding the AES key in the software code. Upon release of the code, however, an attacker can very easily retrieve the key. An implementation secure in this model would look as follows. A key is chosen and a table is made that stores all ciphertexts next to their corresponding plaintexts. Given the block size of the algorithm, it is of course impossible to store such a look-up table in memory. The difficulty that follows from this very strong adversary model leads to designs that are often insecure. Increasing research attention is directed towards the Grey-Box Model.

Grey-Box Model

The third adversary model, and the one we consider in this work, is the Grey-Box model. It lies in between the two extremes spanned by the Black-Box and White-Box models. In this model, an attacker has access to the platform the cryptographic algorithm is implemented on. This model is inherently tied to the physical world a device resides in.

Alternatively, the power consumption or the electromagnetic radiation of a device during encryption was shown to depend on intermediate secret values. By measuring and analyzing these physical parameters, the secret key material can be extracted with surprising ease, at a reasonably low cost, and with basic electronic equipment. Creative new attacks are still being uncovered and include exploiting information from sound, heat, or light. These unintended channels that leak information on key materials are called side channels. It is important to reduce the information leaked through them when deploying cryptosystems in hostile environments (including consumers' hands). In early 2018, the Spectre and Meltdown attacks targeting Intel processors gained widespread media coverage and further stressed the importance and realism of the Grey-Box model.

As the attacker has access to the implementation of the cryptographic algorithm,

there is no limitation in how he uses the device. The attacker can even misuse the cryptosystem in the hope to reveal secret information. By hovering electromagnetic probes (i.e., antennas) over the surface of a decapsulated chip's memory, the stored secret keys could be read out directly. Alternatively, putting the device under stress can trigger an exploitable behavior, e.g., by heating it or by inducing errors in intermediate results using lasers. For cryptographic algorithms to be useful, they will at some point need to be implemented. As a result, these physical attacks are a real concern and their mitigation is critical for the security of embedded devices.

2.4.2 Categorizing Physical Attacks

We need to know what an attacker is capable of in terms of designing a device that is effectively secure against physical attacks. To this end, the Grey-Box adversary model is often refined and subdivided in the following taxonomy of attackers.

- **Class I (clever outsiders)** Attackers in Class I have some knowledge about the subject and generally know what to look for, but have insufficient knowledge of the details of the system. Their equipment is at most moderately sophisticated and they more than often rely on existing weaknesses rather than the creation of new ones.

- **Class II (knowledgeable insiders)** Attackers in Class II have far more specialized technical education and experience than an attacker in Class I. Their understanding of parts of the system can be high and they can potentially even have access to the system. They have highly sophisticated equipment for analysis at their disposal.

- **Class III (funded organizations)** Attackers in Class III are extremely well funded and can assemble teams of experts with complementary skills. They can perform in-depth system analysis and launch sophisticated custom attacks.

They have access to the most advanced equipment. The division of adversaries into classes and the physical nature of the attacks hints at a wide range of existing attacks that can be mounted and leads to further subdivision. An accepted way to partition the attacks is based on the interactivity of the attacker (either passive or active) and the level of intrusion in the system by the attacker (invasive versus noninvasive).

.

2.5 Side Channel Analysis

In practical life, there are many attacks that can be used to reveal the secret key of a cryptographic algorithm, which are known as side channels. For instance, side channels such as the power consumption of the device can often be used to obtain the secret key. You can observe the power profile which has been taken from a micro controller which is typically used to realise several of our products. In these power traces you can see the spike sometimes can be narrow , while other times can be broader. There are three essential requirements to successfully carry out a side-channel attack. These are a perturbation manifestation and an observation. [8] We discuss each of these requirements in detail below.

1. **Perturbation:** A perturbation occurs when the secret that the attacker wants to reveal alters the behavior of the system or its state. Additionally, hardware registers that hold intermediate results in the divider may be perturbed during the computations. For instance, changes in register values are not always visible to a user executing an application. However, the changes are manifested in the power consumption and electromagnetic radiation of the device. The instructions executed by the program are manifested in the execution time as

well as the radiation and power consumption. These manifestations provide information about program's internal hardware.

2. **Manifestation:** Most often the perturbations in the system, brought about by the execution, cannot be directly accessed by an attacker. In many cases however, they are manifested through side channels, which can generally be monitored by using a small resistor R in series between V_{dd} or V_{ss}. Using this property the key of a cryptographic algorithm can be inferred from the power consumption statistics.

3. **Observation:** An attacker would have to observe the side channels in order to obtain the required information about the secret. Timing based side channels require monitoring of execution time. The clock used to measure time would need to be highly precise in order to distinguish between the micro architectural events in the execution. In certain cases the attacker creates an environment which forces the program or system to behave in a specified way.

2.5.1 Power Attacks

Every digital circuit built today is based on Complementary Metal Oxide Semiconductor(CMOS) technology. Power consumption of CMOS is composed of dynamic and static power consumption. Dynamic power consumption directly relates to the data of circuit processing.

Most of the cryptographic devices are implemented by TTL or Transistor Transistor Logic. The cipher operation is realized through the state change of the logic gate circuits, and the state of the gate circuit changes through the current change. Next, we will analyze the state change of CMOS inverter to illustrate the principle of power analysis. The capacitor gets discharged in this state. In a CMOS, when V_{in} is high voltage of Logic"1" then NMOS conducts and V_{out} is Logical "0". Whereas

when V_{in} is low voltage or has Logic "0" then the load capacitance gets charged

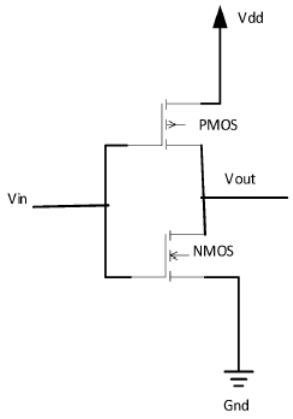

Figure 2: CMOS Circuit Diagram.

Because of the fluctuation of V_{in} the difference of power consumption occurs in the circuit due to input transitions from 0 to 1 and vice versa. Because of the difference in power values that result from these transitions, we get the basis for differential power attack.

Simple Power Analysis is the kind of attack that directly interprets the power consumption of the device based on the operation taking place at a particular time instant. A set of [8] power consumption of the device. These attacks are noninvasive in nature. Differential Power Analysis exploits the fact that the power consumption of the same operation at different instants of time depends on the data being processed. Even if you protect the design with SPA, DPA will still work by modifying the data. Your system is also largely dependent on what data is being used for operation. If you get the idea of the data being used, the key can be revealed. Attacking the output of Mix-Columns is very costly as the function is defined for 32 bits. This means the attack requires a key hypothesis for 2^{32}.

Differential Power Analysis

We will assume that power leakage follows Hamming weight since DPA depends

on the underlying data and not the algorithm itself. As we can see, we have varied s from 0000 to 1111. Then we list the Hamming weight for each number and the LSB. The relationship between power and Hamming weight can be defined as

$$P \propto aHW(s), \tag{1}$$

where a is the proportionality constant, and HW(s) is the hamming weight for a particular byte.

We divide the number into two categories 0 LSB & 1 LSB. Both categories have 8 numbers and if we take the difference of the mean it comes to 1. However, if we increase the number of samples and plot the DoM mean, we get to know that the difference of the mean will be close to 0. It must be noted that S-box of the first round is targeted by the attacker since that is the only function in AES in which the data and cipher keys enter a direct operation. These predictions are correlated with the real side-channel output. This correlation can be measured using the Pearson correlation coefficient which is as follows

$$C(T, P) = E(T.P) - E(T)E(P)/\sqrt{Var(T).Var(P)}. \tag{2}$$

Here *E(T)* denotes the average trace and *Var(T)* denotes the variance of the set of traces.

A sample DPA test is as follows at first, N plaintexts are randomly generated. Power consumption measures are taken for each plaintext. The attacker gets N measures with each containing n samples. Then, a hypothetical model of the AES. is fed with the plaintext and one byte of the first subkey. To this output hypothesis, a selection function D is applied. This selection function divides the measures in two sets. One where the selection function returns 1, and the other for the return 0.

For each set, the average is computed. Then, the difference between the two averages is calculated. For each differential curve, the highest peak and the mean value of the curve were calculated. These two values are then divided. The above steps are repeated for each of the hypotheses. This leads to 2^8 differential curves. These differential curves are now inspected.

2.5.2 Power Model

Its a vital aspect of power attack through which we can estimate the corresponding power consumption. Power analysis largely depends on the simulation of power, which predicts the change in the power values with the change in the internal values. Power model is the bridge between both values.

While calculating the power values, relative power is more important rather than the actual values themselves to understand the correlation between the change in the micro controller related values with trends in power consumption. There are two models which are commonly used: Hamming weight and Hamming distance.

Hamming Weight In this power model, the power consumption is considered as a function of the hamming weight of the registers inside the circuit at a particular time t. It doesn't depend on the past outputs of the voltage of CMOS present in the registers. Hamming weight is the number of 1's present in an n-bit binary number.

Hamming distance Power consumption depends on the Hamming distance between the previous and the next state. This implies that in this case the power consumption is a function of time. $P = HD(V_0, V_1) = HW(V_0 \bigoplus V_1)$ where V_0 and V_1 are the voltages of the register when it transitions from $V_0 \rightarrow V_1$.

2.5.3 Differential Power Attack

We observe the power traces of atleast 1000 AES operations occuring in the hardware and assume a tuple of the form of ($Ciphertext_i$Curve) Curve represents the power trace that we are considering for the DPA procedure. We assume a key for the last round of keys and the actual key has a dimension of 128 bits and there are 16 Sboxes with the last Sbox getting targeted. A byte of a key is guessed and the last round of Sbox is observed that we are targeting. Each one of the 8 bits can take either a 1 or 0 value, so we put the power trace in two different bins or divide the measurements into two groups based on the output of 0 and 1 and take the average of the two groups. The difference of the two curves, if your key assumption is correct, then we will get a nonzero spike only for the Sbox you are observing, whereas for the wrong key guesses you get close to zero power trace. This property helps us in revealing the key once we repeat these steps for other Sboxes.

There are two scenarios that can occur : in one instance you can wrongly assume the key and then in that case the probability of guessing the correct key bits is less than half, but if the key is correctly guessed then in that case and the output acts like a pseudo-random function. We do not require any knowledge of the plain text for successfully attacking the cipher.

We denote the data in the form of a table which has had cipher text that you are testing in the rows and the sample points of the power trace in the columns to represent the trace in a tabular fashion. Let us take the case of AES to understand how this attack works by considering a particular bit of AES. For a particular round of AES, a circular shift operation is applied, and then the row is XOR'd with the key to get the cipher bit.

We take the inverse of the bit and take the XOR with a an assumed key bit and compare it with values of the value in the original state matrix. Based on the value

of the LSB, we put the values in separate groups and repeat this process for at least 100000 data points of the trace.

If the guesses are wrong, then the function will act like a pseudo-random function, in which case the difference of the mean will be close to zero, whereas for the correct bit the average will be close to 1. The difference of mean for a particular bit, ciphertext and key byte can be calculated using

$$\Delta_D = \frac{\sum_{i=1}^{m} D(C_i, b, K_s) T_i[j]}{\sum_{i=1}^{m} D(C_i, b, K_s)} - \frac{\sum_{i=1}^{m} (1 - D(C_i, b, K_s)) T_i[j]}{\sum_{i=1}^{m} (1 - D(C_i, b, K_s))}, \tag{3}$$

where Δ_D is difference of mean, C_i, b, K_s are the corresponding ciphertext, bit and keybyte for which the difference of mean is being computed.

The first fraction represents the average of the sums for 1-bit groups and the other fraction represents the average for 0-bit group. If our guess is incorrect, then Δ_D will tend to zero, whereas if our guess is correct, then the value it will tend to 1 and this is what reveals the key in the end.

2.5.4 Correlation Power Attacks

Like the difference of mean-based DPA attacks, correlation power attacks also target the intermediate values of the AES, which are computed with the guessed key, subbyte, and cipher text. From these values we get a hypothetical power trace for particular values of plain text given to the cipher. The actual power value for the ciphertext value is also noted in the form of a matrix. We compare it with the actual power trace and store these values in a correlation matrix. It is later noted that one of the columns of the hypothetical matrix corresponds to the actual key. Thus, the role of the attacker is to look for the similarity between both matrices.

We choose either Hamming Distance or Hamming Weight Model to simulate a hypothetical power model which is stored in the form of a 2-D array trace[time][Power

Trace]. Each point in the matrix then describes the power sample value for the corresponding value of time. For the hypothetical power consumption, the attacker obtains the hypothetical power matrix is obtained by starting from the ciphertext. The attacker targets a specific key, byte key, and finds out the corresponding byte in the ciphertext, which shows the last round toggling in the registers storing the state of AES during the computation. One may note from the table shown below that due to the shift row operation, the Hamming Distance computations should be properly expressed for the iterated AES architecture. For example, note the toggling in the register R1, can be found by HD(S1 , S(S13)$\hat{\text{k}}$1), where HD denotes the Hamming Distance of two bytes. While the second term in the HD function is directly available to the attacker from the ciphertext byte denoted by Cipher=C1 , for the first term the attacker needs to guess a key byte. The attacker thus guesses the key byte, k5 and performs XOR with the corresponding ciphertext byte, and performs Inverse SubByte to obtain the state S1. The attacker thus develops the hypothetical power values by obtaining the Hamming Distance between the cipher and Inverse SubByte of SCipher $\hat{\text{key}}$. We try to find the correlation between the i^{th} column of the hypothetical power matrix and j^{th} column of the actual power trace using the following formula

$$Result[i][j] = \frac{\Sigma_{K=0}^{NSample}(h(P[i][k]) - \overline{H[i]})(T[j][k] - \overline{T[j]})}{\Sigma_{K=0}^{N}(P[i][k] - \overline{H[i]})^2 \Sigma_{K=0}^{N}(hP[i][k] - \overline{H[i]})^2}, \tag{4}$$

where we denote the candidate key and j represents the time instant in the power trace when the correlation is computed. After a sufficient number of traces, a significant peak is expected, which reveals the subkey byte of AES.

2.5.5 Template-Based DPA Attacks

Power attacks from the previous sections assume that the attacker has a very restricted knowledge of the attacked device. We used very simple statistical models

for attacking the device like the difference of mean and correlation between the power traces and the Hamming weight of the power trace. However, to target the device more strongly, we need a more [8] intimate knowledge of the device power model.

In this section we assume that the attacker has the chance to characterise the power consumption of the device based on templates. This is a build-up for template DPA attacks, the strongest side channel attacks. The number of traces required to break the AES is very small as the template narrows down the relevant power traces related to the actual key. To choose the optimal points of interest, there is no fixed strategy.

We consider the probability of the trace and employ *Bayes Theorem* to calculate the probability such that if the key is k_j considering different trace values. The probability can be be determined using the following formula

$$p(k_j|t_j) = \frac{p(t_i|k_j).p(k_j)}{\Sigma_{l=1}^{K}(p(t_i|k_l)).p(k_l)}, \tag{5}$$

for j=1,.....K.

The prior probabilities are the probabilities for the different keys without considering the traces t_i. Through this theorem, we catalogue the trace for a particular key guess. This approach is called as the maximum likelihood approach. Since the power and traces are statistically independent, we can multiply the corresponding to different traces

$$p(k_j|T) = \frac{(\Pi_{i=1}^{D}p(t_i|k_j)).p(k_j)}{\Sigma_{l=1}^{K}(\Pi_{i=1}^{D}(p(t_i|k_l))).p(k_l)}, \tag{6}$$

for i=1,...,D and j=1,...,K.

After having built templates for the attacked intermediate result, we collect the actual power traces from random plain text and perform DPA while varying the values of i=1,...,D and j=1,...,K. We then compare it with the template created relating the power traces with the corresponding Hamming weights and we get the corresponding

key.

2.5.6 Fault Attacks

Fault analysis is another kind of side channel attack, it is a malicious aberration in the normal execution of the target cryptographic algorithm. This aberration often leads to additional information leakage, which can then be used to try and recover the key. The complexity of injecting a fault and subsequently recovering the key depends on a number of factors, including: the nature of the fault, the spatio-temporal characteristics of the fault, and the fault propagation characteristics of the target algorithm. Examples of fault nature include bit flips, byte faults, stuck-at faults, or random faults. The characteristics of fault attacks depend on both the block cipher and the precise location the fault is injected. The precise timing needs to be extremely accurate or we can end up with the wrong key. Based on the attack principle, the attacks can be categorised into the following major categories:

Differential Fault Analysis

DFA is a fault analysis technique in which an adversary injects a fault with a spatio-temporal characteristic and then analyzes the difference of fault-free and faulty cipher-text pairs to recover the secret key. Previous instances of differential fault analysis have allowed the complete recovery of keys in the case of AES 128 cryptography algorithm by using a single fault injection in the system. The practicality of the differential fault analysis depends on the number of faulty cipher texts required for the attack. Various fault models can be employed to execute differential fault analysis

- **Bit Model** This fault model assumes that only a single bit is affected during the fault injection. The attack is not practical since with a random fluctuation the wrong bit can get.

Fault sensitivity analysis

FSA exploits the data-dependent transition point between the fault-free and faulty behavior of a given target cipher implementation. For example, adversaries may discern the critical point of fault occurrence while gradually increasing the intensity of the fault injection. FSA then exploits the relationship between the fault sensitivity and the processed sensitive data to retrieve the secret information from a cryptographic device.

Differential Fault Intensity Analysis

DFIA represents a class of fault attacks that combine the principles of side channel analysis techniques such as DPA with that of fault analysis for key recovery. DFIA requires only faulty ciphertexts and chooses the key hypothesis that can be traced back to a faulty intermediate state value with a highly biased distribution, as opposed to a uniformly random distribution. DFIA requires a higher number of fault models owing to its inherently statistical nature; however, it is a potent threat to the security of several block ciphers.

2.6 Related Works

The first set of power attacks were revealed in the late nineties by Paul Kocher, et al., [7]. The findings pointed out the relationship between power and cryptographic implementation. The power and electromagnetic radiation of the device during cryptographic implementation. By analysing and comparing the power traces, cryptographic keys are extracted easily. New and creative ways are being uncovered by the attackers to reveal the information from acoustic light, heat side channels of the device.

Countermeasures [1] become very essential to counter these attacks. Countermeasure techniques are costly as it can decrease the speed of the device. Therefore, we need countermeasures that while making the device more secure, it also does not effect

the efficiency of the device. For example, adding random delays in the device can slow down the speed of operation considerably and may not be effective for other kinds of attacks apart from differential power attacks, such as correlation power attack and differential fault attacks. Jean-Sebastien Coron and Ilya Kizhvatov proposed a random delay operation method for countering differential power attacks in embedded software on a 8-bit AVR platform. Most attacks require multiple traces to become asynchronous so that the peak values which were being used to reveal the key become random in nature and an incorrect key is revealed. Similar work was done by Clavier et al., [5] which shows that the number of traces for a successful DPA attack grows with the increase of random delay. These attacks were later improved by Benoit and Turnstall to increase the variance of the sum of delays and decrease the difference of mean of the actual peaks.

The other countermeasures which were employed specifically for CPA used a mixture of random clocks & delays or random switching logic bus invert coding techniques. The random switching logic was used for stream ciphers for Encoro-128 [10]. By adding noise to the clock and voltage, we can randomise both the signals and hence mask the data preventing it from getting revealed during the attacks. The phase-shifted clocks are fed into a series of clock multiplexers and use random number generators to switch between input clocks and the MUX. While switching between the input clocks, the MUX holds the selection of clock until the next output cycle clock is equal to the new clock we get from the multiplexers. This produces inherent delay and thus counters the AES-128 implementation. The second countermeasure uses linear feedback shift registers. By varying the values of the number of rings and shift registers, the voltage noise can be varied and hence hide the voltage noise better.

Kean Hong Boey et al., [1] talks about varying the clock dynamically to insert

dummy operations during the idling clock cycles or in other words, decreasing the signal to noise ratio to counter DPA. The main premise of the paper is the misalignment of power traces that help us obtain the peaks at the right time. The randomisation is achieved by dynamic frequency switching or random delay insertion. The random delay is controlled with a linear feedback shift register to create temporary misalignment in power traces.

In a more recent paper, a similar technique was adopted by Benjamin Hetwer et al., [6] for lightweight side channel protection. Instead of using an LFSR, the runtime configuration of the FPGAs is exploited to produce a highly unstable clock signal for 20 million different execution times for an AES encryption by using a new protecting method called mixed mode clock manager on a Xilinx ZYNQ Ultrascale FPGA to generate more than 2000 distinct frequencies.

2.6.1 Experimental setup for DPA attack on SRAM-based Xilinx Spartan-II

We will look at the hardware implementation done by Mehdi Masoomi, Massoud Masoumi, Mahmoud Ahmadian [9] The measurement setup consists of a Xilinx Spartan- II FPGA board, an Agilent MSO-7034A sampling oscilloscope with 2 Gs/sec, BW = 350 MHz, current probe Agilent 10073C to measure the supply current. The board uses two separate power supplies, a 3.3V supply for I/O and a 2.5V supply for the core cells. Using a small resistor inserted between the FPGA board and the power supply, the power consumed was measured.

The plaintexts were generated from a 128-bit LFSR on the FPGA. To reduce switching noise, all measurements were averaged over ten times. The working frequency of FPGA board was lowered to 1KHz by using an internal clock divider (the actual frequency was 10 MHz), to reduce the switching noise and providing more

data to improve the accuracy of the attack. For implementing the correlation analysis, we let the FPGA encrypt the same N plaintexts with the same key. While the chip is operating, we measure the power consumption for the first clock cycle of each encryption, i.e., the cycle in which S-box operation is performed. Then, the power consumption trace of each encryption is averaged 10 times to remove the noise from our measurements, and we store the maximum value of each encryption cycle so that we produce a columnar matrix with the power consumption values for all texts. We denote it as the global consumption matrix.

3 SIDE CHANNEL ATTACK IMPLEMENTATION

In this section we will be explaining my experimental setup for implementing and later attacking the AES-128 implementation.

3.1 Experiment Setup

We use state–of–the–art Chipwhisperer Lite ARM and Chipwhisperer Nano from NEWAe Technology inc. for their ease of use and low cost to implement my experiments. It is an open source hardware for implementing side channel attacks. The board is divided into parts : capture and victim hardware. Capture hardware is used for attacking the AES implementation and victim hardware for implementing AES-128. Chipwhisperer Nano [13] was my introduction to capture hardware. It has a small subset of features compared to Chipwhisperer Lite and has the following features:

- ADC is capable of sampling up to 20 MS/s, using either an external clock (synchronous device) or an internal clock (both synchronous and asynchronous).
- ADC hardware trigger uses rising-edge input and starts sampling with the first device clock after the trigger line goes high samples of user-configurable length.
- STM32F030 targets for loading example code onto, including a programmer built into the ChipWhisperer Nano. It also has 16KB of flash memory, which is not enough to run MBEDTLS RSA. This microcontroller is available with larger flash sizes, so if you want to run the RSA glitch attack, you will need to replace the microcontroller on the board with one of the ones with more flash.
- Crowbar based VCC glitching, approx 10ns resolution on glitch width and offset (glitch offset from trigger with up to 200 ns jitter).

The Chipwhisperer nano has limitations regarding clock glitching during Fault attacks. To cover that, we use CW1173 Chipwhisperer lite ARM. It has the following features:

- Synchronous (capture board and target board both use the same clock) capture and glitch architecture, offering vastly improved performance over a typical asynchronous oscilloscope setup 10-bit 105MS/s ADC for capturing power traces
- Can be clocked at the same clock speed as the target and 4 times faster +55dB adjustable low noise gain, allowing the Lite to easily measure small signals.
- Clock and voltage fault generation via FPGA-based pulse generation XMEGA (PDI), AVR (ISP), and STM32F (UART Serial) bootloader built in

We used the openADC for capturing the trace while running AES algorithm on STM32F0 8-bit microcontroller targets. We used Anaconda Jupyter Notebook for statistical analysis of the trace and GNU ARM C library for the AES implementation. We use Chipwhisperer software [12] for running AES and implementing the attacks. We connect the board to the PC and run the firmware for building and establishing connection with the target. The serial baud rate used is 38400 and later program the target to run multiple iterations of AES. Later, we run the code and capture the corresponding power trace of the device.

3.2 Capturing Power Traces Using Firmware

Measuring the power consumption of the target during sensitive operations can allow you to determine if the target is leaking information about its sensitive operation (such as encryption). Analysis of the power consumption may allow you to recover the secret that should have been inaccessible inside the target (such as the encryption key).

Algorithm 1: Basic Trace Capture.

```
Result: Power Trace Captured
reset target;
while Number of characters > 0 do
    read the password;
    add delay of 1 ms;
end
target writes the guessed password;
scope captures the power;
if return value then
    timeout happens;
end
get the last trace from the scope;
return the trace;
```

3.3 Power Analysis For Revealing Password

We do a basic password bypass in which we store a basic password "hello" and capture the corresponding power trace while the password is being serially input after activating the scope using the function call scope.arm().

Algorithm 2: Basic Password Bypass.

```
Result: Password is Bypassed
Set threshold value;
while chosen character index<total character length do
    capture the reference trace;
    for character in alphabets do
        capture the trace of the guessed character;
        calculate the difference between the character and reference trace;
        if difference>threshold then
            print the character;
        end
    end
    iterate the character index by 1;
end
```

We use this algorithm later to compare the reference trace acquired while reading with that of other alphabetical characters. Moreover, it needs to be made sure to keep the threshold value reasonable as well. Otherwise, the serial buffer overflows.

3.3.1 Results

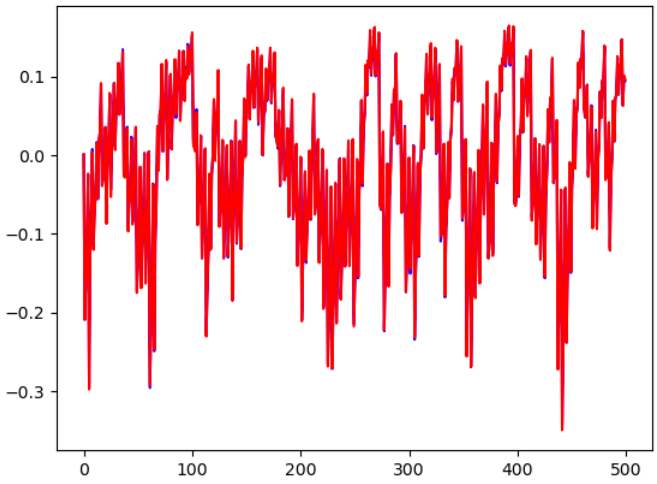

Figure 3: Power trace comparison between reference trace and “A”.

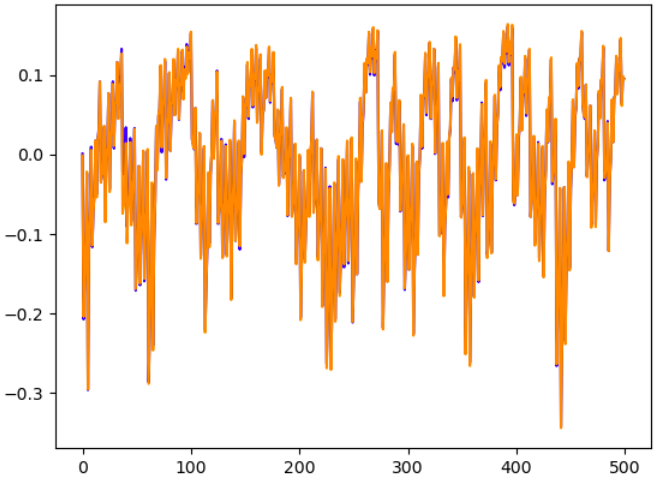

Figure 4: Power trace comparison between reference trace and“J”.

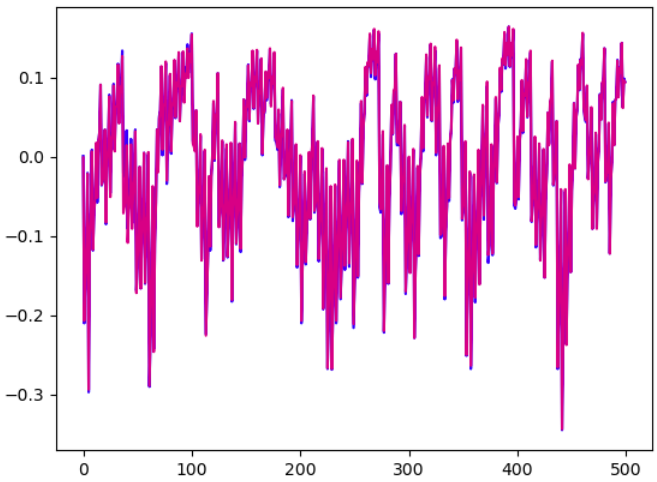

Figure 5: Power trace comparison between reference trace and “N”.

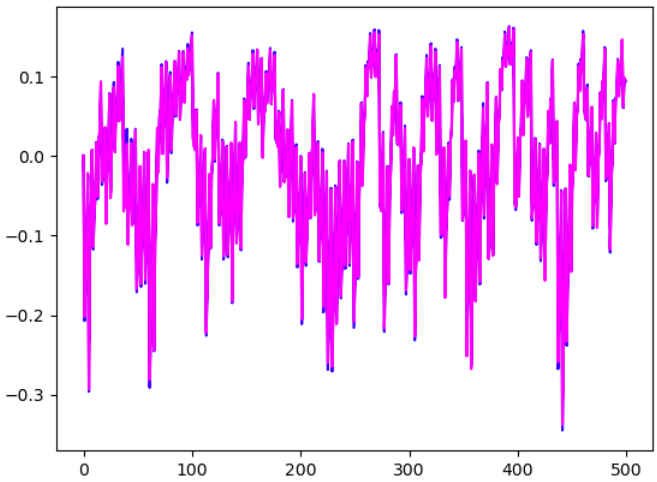

Figure 6: Power trace comparison between reference trace and “Q”.

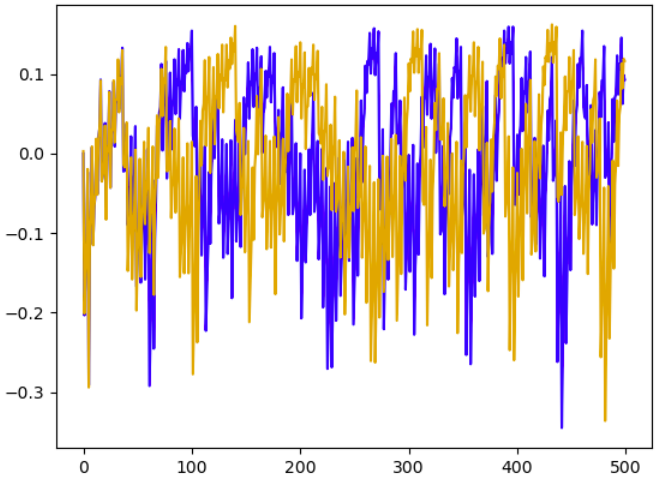

Figure 7: Power trace comparison between reference trace and “H”.

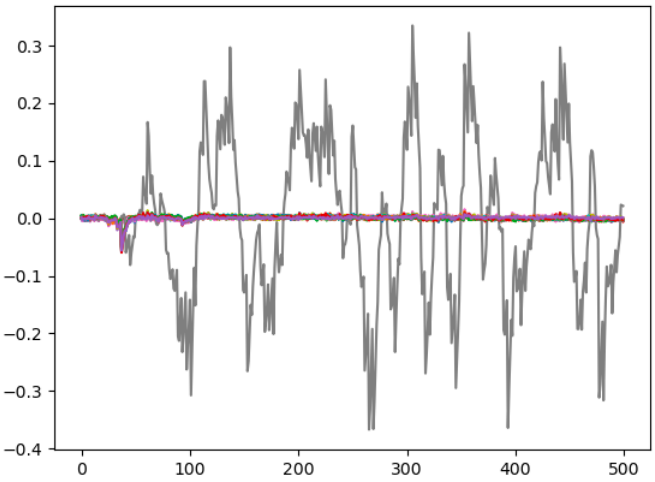

Figure 8: Power trace comparison for correct guess vs all other alphabet guesses.

3.3.2 Conclusion

The attack is based on different code execution path relationships with the power traces that are obtained from them.

You can observe in the first 4 results that the power trace for random alphabets overlaps with that of the reference power trace, implying that any of these alphabets are not the first character in the password. However, we see a significant difference when we compare the power trace for writing "h" in Figure 5. Thus, we can conclude from this that "h" is the first letter of our password. The difference of trace for h can be seen very vividly when we compare it with all other alphabet traces in Figure 6. This technique of revealing the password is used again during differential power attacks as well.

3.4 Relationship Between Power And Hamming Weight

In the previous section, we analysed the microcontroller power and its relation to the different set of instructions being written to it. Using that concept, we were able to reveal the text that was written into the microcontroller.

In this section we will be looking into not how not only the different instructions but the data that we are writing to the microcontroller also have effect on the power consumption. We will be specifically seeing the effect of Hamming weight on power consumption in this experiment.

Algorithm 3: Calculating Hamming Weight Swings.

```
Result: Hamming Weight Swings Plotted
Read 16 byte random plain text;
read 16 byte fixed key;
Write fixed key;
for Traces in the range of (0,100) do
    if plaintext[0] & 01 ==1 then
        last byte of plain text = 0xFF;
    else
        last byte of plain text = 0x00
    end
    Write random plain text;
    capture and write the power trace and corresponding plain text in trace
     array;
end
for text in trace array do
    if text[0]==0x00 then
        append text in list1;
    else
        append text in list0;
    end
    iterate the character index by 1;
end
calculate the average of list1;
calculate the average of list0;
calculate the difference between average of list1 and list0;
plot the difference;
```

3.4.1 Results

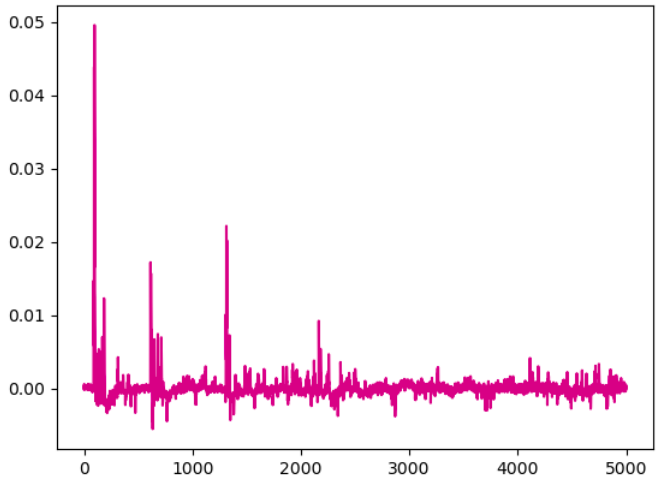

Figure 9: 8-bit hamming weight swing power trace.

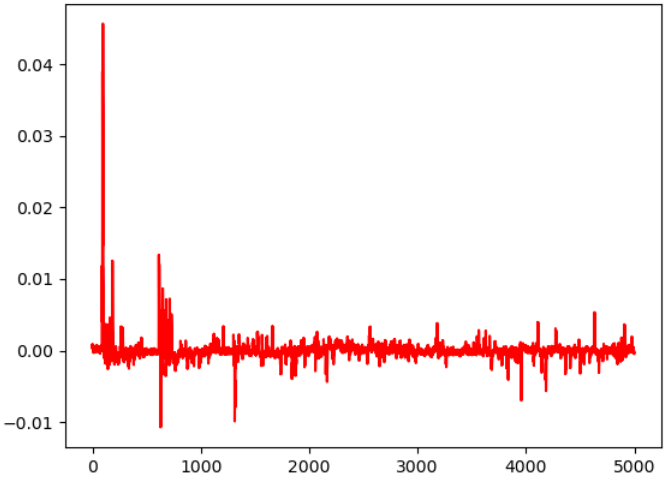

Figure 10: 7-bit hamming weight swing power trace.

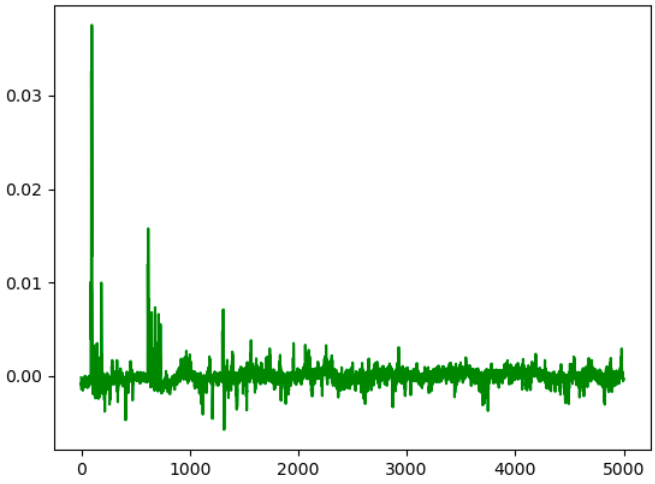

Figure 11: 6-bit hamming weight swing power trace.

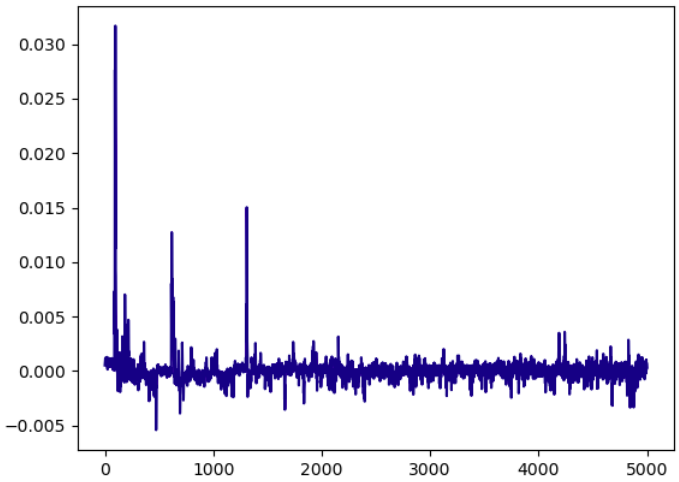

Figure 12: 5-bit hamming weight swing power trace.

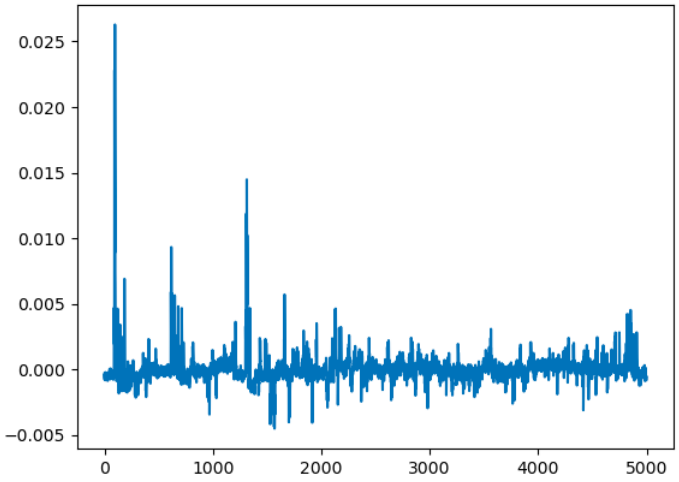

Figure 13: 4-bit hamming weight swing power trace.

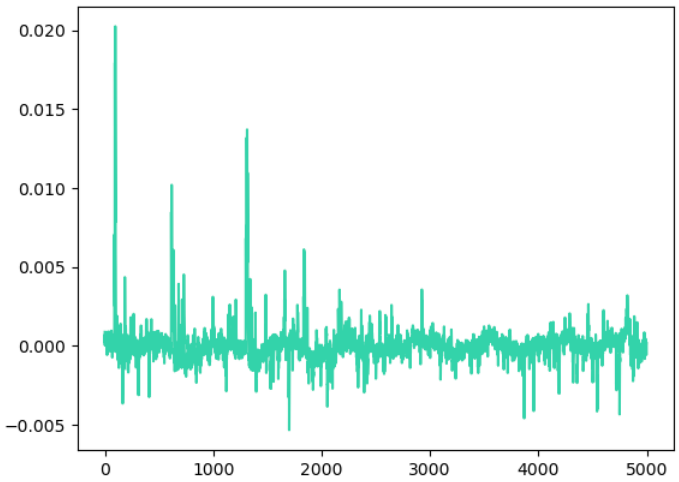

Figure 14: 3-bit-hamming weight swing power trace.

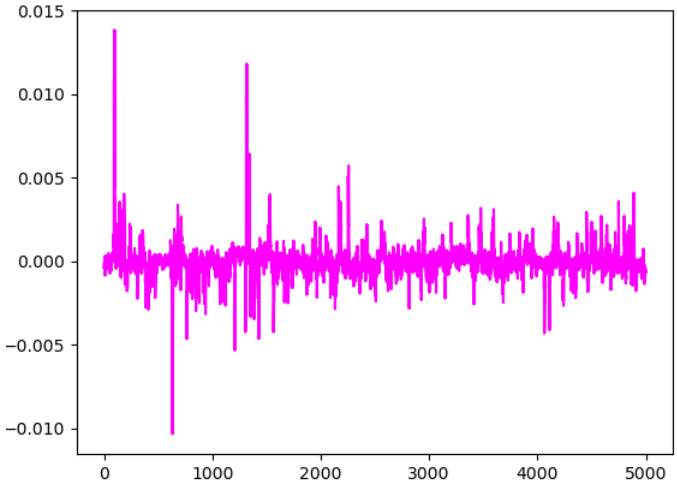

Figure 15: 2-bit-hamming weight swing power trace.

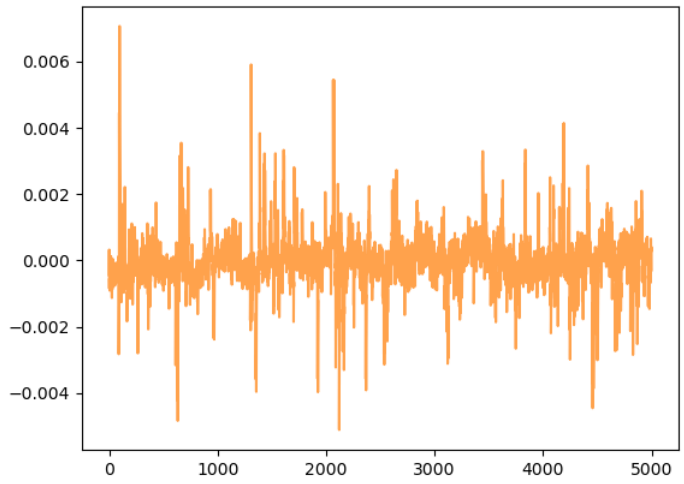

Figure 16: 1-bit-hamming weight swing power trace.

3.4.2 Conclusion

We can observe in the maximum power trace as we decrease the Hamming weight. The 8-bit hamming weight number swings the highest at 0.050 to around 0.006 for hamming weight 1. This shows that Hamming weight of the data being passed can be used to differentiate and thus decipher in some way what data has been written by observing the power trace of Hamming weight swing.

3.5 Recovering AES from a single bit of data

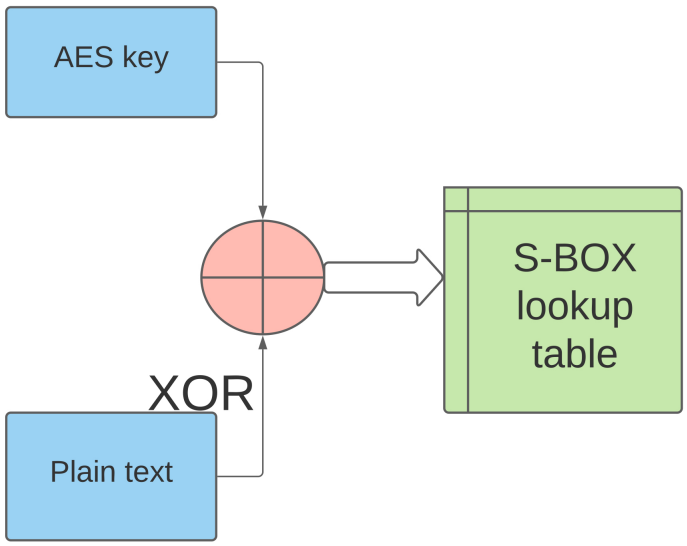

Figure 17: AES Sbox.

AES Sbox

Sbox comprises of the byte substitution layer of the AES algorithm, which comprises of a 16x16 matrix that is the nonlinear element of the AES implementation. We input the XOR of plain text and key combination into the S-Box lookup table as an index value, and we get the corresponding value of that index. Side channel attacks the internal values of AES encryption. Lets assume that while encrypting the data, one bit of data gets exposed when the attacker is probing the chip. We create a list of random 1000 numbers ranging from 0 to 255. That can be assumed as the random input data that is being input to our device. We set the value of the secret key byte to be 0xEF and store the data in our chip which is being attacked.

Leaked Data

For obtaining the leaked data, we will input random 8-bit data and observe any single bit. In my experiment, we observe the last bit by masking the data with 0x01 for all input bytes. We get a random sequence of 1's and 0's as shown below:

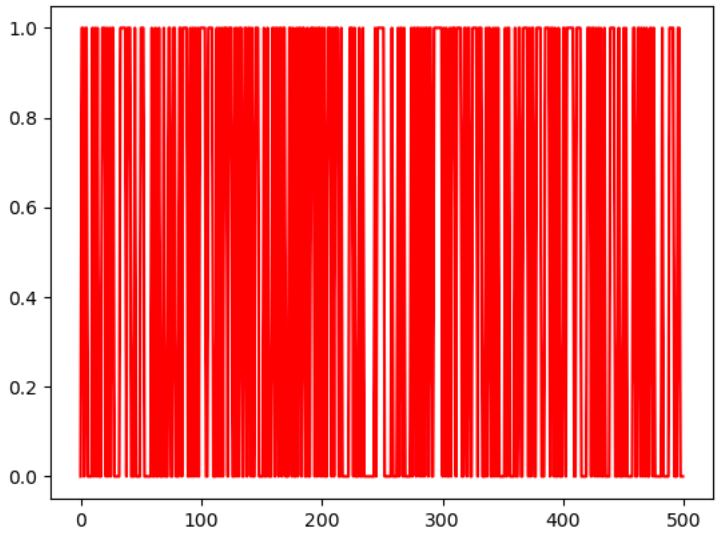

Figure 18: Leaked Data Of Random 500 Bytes.

Hypothetical Leakage & Bitwise Key Guess

Now that we have got the leaked data by masking the last bit of all input data, next we will create a simulated leakage or hypothetical leakage by masking the internal value we obtain from the Sbox. Later we compare both leaked data and hypothetical leakage to count the number of elements, i.e., the same values and index positions. We use the same technique to guess the key byte by considering all bit positions of the 8-bit input data.

We calculate the hypothetical leakage value for each key byte from the range of 0 to 255, and calculate the number of bits between the actual and hypothetical leakage and observe the results.

We repeat the same experiment for all other bit positions for plain text and key and measure the number of correct bits. We have discussed all steps in the algorithm.

Algorithm 4: Key Reveal From Bitwise Guessing Loop.

```
Result: Key byte from 0 bit guessed
create sbox;
store secret key byte;
generate random input 8-bit data in the range(0,255);
for data in input data do
    leaked data append( data[0]&0x01);
end
for guessed bit in range (0,8) do
    for key guess in range(0,256) do
        for input byte in input data array do
            Hypothetical leaked data = sbox[key guess ⊕ input byte] & 0x01;
            count number of data bits at the same index position for leaked
             data and hypothetical leaked data;
            plot the results;
        end
    end
end
```

3.5.1 Results

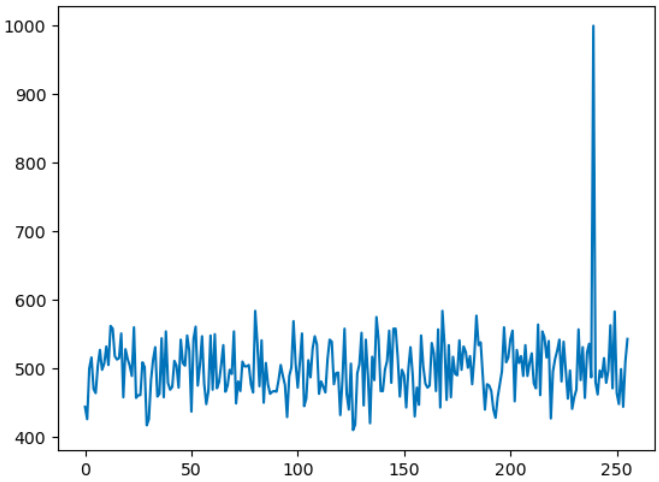

Figure 19: Attacking 0^{th} bit of the input data for all the key guesses to reveal the secret key EF.

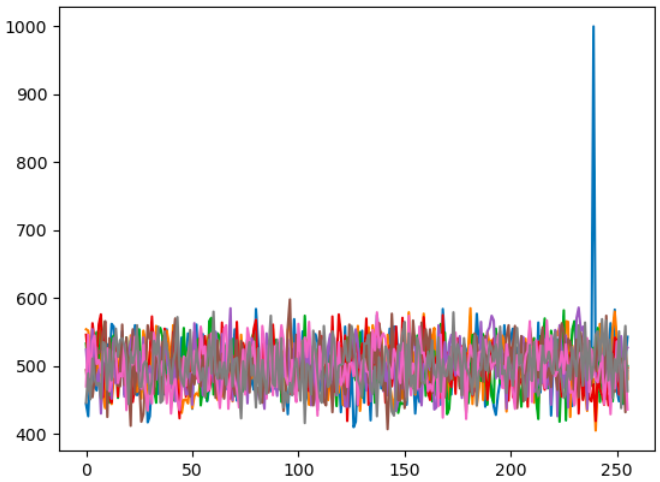

Figure 20: Attacking all the bits for all input data and guessed key combinations to reveal the key as EF.

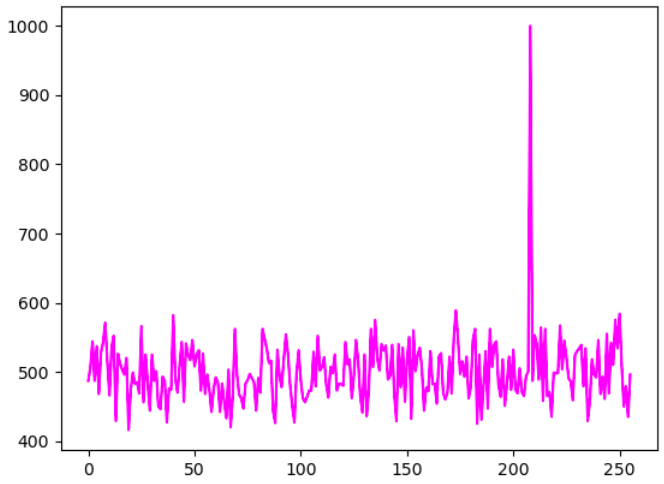

Figure 21: Attacking 0^{th} bit of the input data for all the key guesses to reveal the secret key D0.

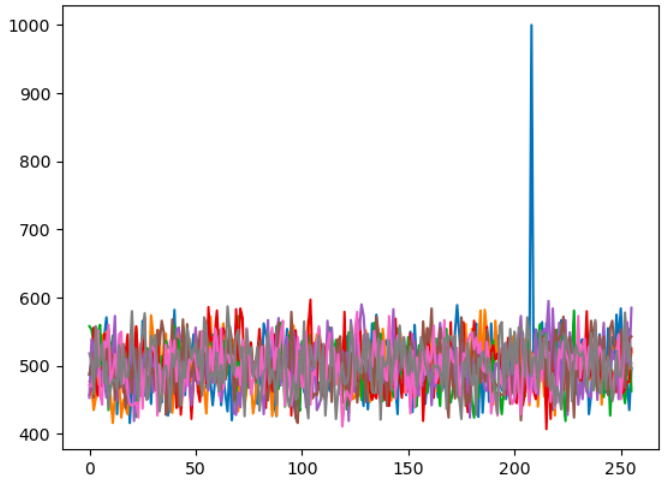

Figure 22: Attacking all the bits for all input data and guessed key combinations to reveal the key as D0.

3.5.2 Conclusion

In figure 18 & 19 we observe that there is a relationship between the number of matches between the hypothetical & leaked data and the actual key. Only for the correct value of the key do we get the full number of matching bits. For all other key guesses, the number of matches is more or less the same, meaning that those are not the correct values of the keys. We can conclude from this experiment that the attacker can know the exact key with just a single bit leakage.

3.6 Differential Power Attacks On Firmware Implementation Of AES

In the previous sections, we learnt about the behaviour of power consumption with different types of data that are being written into the target device. In the first two sections we saw the role of data and the hamming weight of the data being used. The power consumption being used in these devices helps us deduce the data that is written to the device. In the last section, we see how even the leakage of a single bit can reveal the secret key when comparing the hypothetical power leakage with the actual leakage of a single bit in the input data byte. In this section, instead of using a single bit value, we will be using power analysis.

Differential Power Attacks Use power analysis to choose a target bit and then perform the difference of mean attacks. We follow the following steps to perform these attacks.

- Calculate the power trace while writing from at least 1000 plain texts into the device keeping the 16 byte key constant.
- Calculate the intermediate values using plain text , key and sbox and calculate the hypothetical leakage for all guessed key bytes from 0 to 255.

- Choose a target bit and mask the hypothetical leakage. Based on the value obtained, place the corresponding trace array vectors in separate arrays as list_one and list_zero for each iteration of the key byte guess.
- Calculate the average separately for the elements in list_one and list_zero.
- Compute the mean difference of the values, and store in an array for each corresponding value of the key guess.

3.6.1 Optimum Operating Conditions

We need to set the conditions for openADC before beginning the experiment for optimum conditions to perform DPA attacks. We require these conditions to capture the trace when we are running the AES on multiple random plain text. We capture 5000 samples per plain text. We perform the experiment for 100 traces and then increase the traces by 1000 traces each time and observe the results until we reach 5000 traces. The value of stored key **9E E3 2D BD 4A F0 FF 69 25 0B 53 9D 3A 02 08 C1** is used to create ciphers from random plain text.

Table 2: Operating Conditions Of Open ADC For DPA.

ADC Parameters	**Values**
ADC clock source	clkgenx4
ADC clock phase	0
ADC clock frequency	29538459
ADC baud rate	29538459.0
ADC gain	24.8359375 dB
ADC locked	True
ADC frequency counter source	0
ADC frequency counter source c	external clock
ADC clkgen source	system
ADC external clock frequency	10000000
ADC clkgen multiplier	2
ADC clkgen divider	26
ADC clkgen frequency	7384615.384615385
ADC clkgen locked	True

3.6.2 Attack

Algorithm 5: Differential Power Attacks.

```
Result: Key Byte Guessed
create sbox;
set 16 byte key;
generate random 16 byte random plain text;
for key guess in range(0,256) do
    for trace_index in range(0,len(trace_array) do
        hypothetical_leakage = sbox[key_guess ⊕ input_byte];
        if hypothetical_leakage & 0x01 == 1 then
            one_list.append(trace_array[trace_index])
        else
            zero_list.append(trace_array[trace_index])
        end
    end
    one_avg=mean(one_list) zero_avg=mean(zero_list) difference_of_mean =
     maximum(one_avg-zero_avg)
end
```

To perform the attack, we choose a subkey to attack. We consider the guessed key bytes in the range of 0 to 255 and calculate the hypothetical leakage for the internal value of AES computed. We mask the last bit of the leakage and place the trace array in two separate arrays: one list and zero list. After that, we compute the maximum mean difference between the average of one list and zero list and plot the final result.

3.6.3 Results

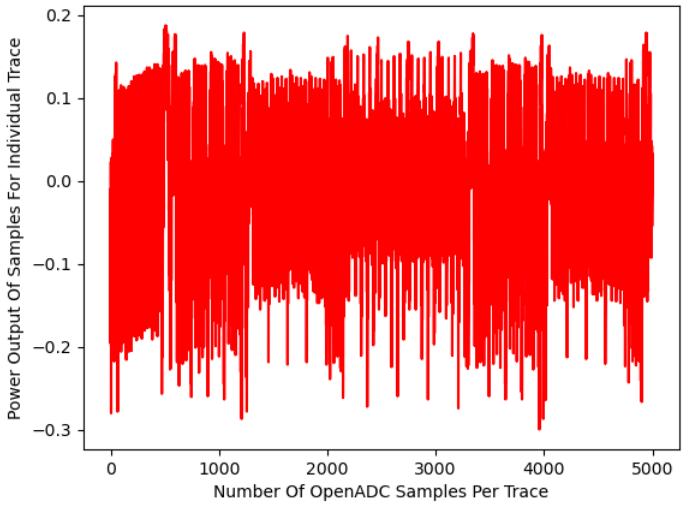

Figure 23: Power Trace Of A Plain Text Iteration.

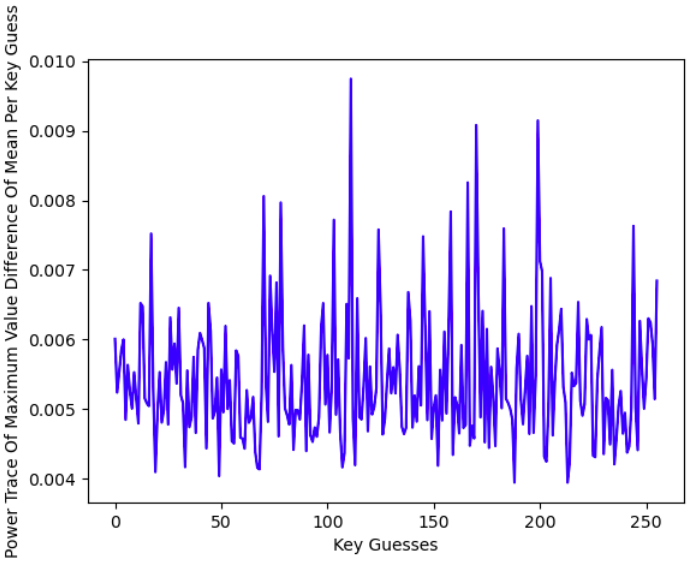

Figure 24: DPA on 0^{th} Subkey Performed For 100 Traces.

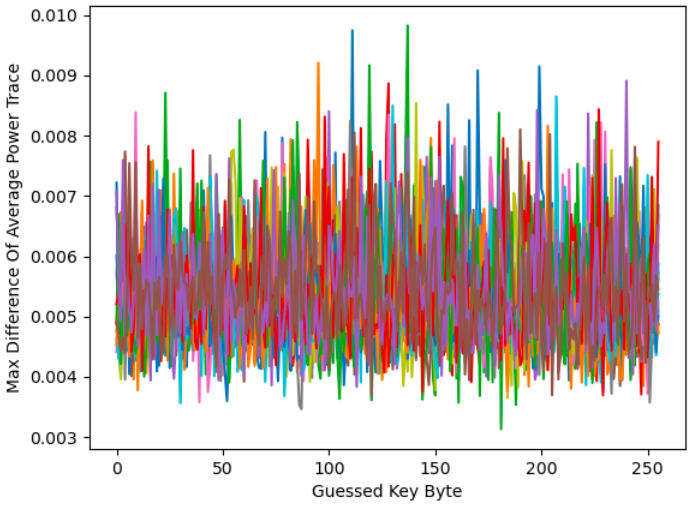

Figure 25: DPA for all Subkeys guesses performed For 100 Traces.

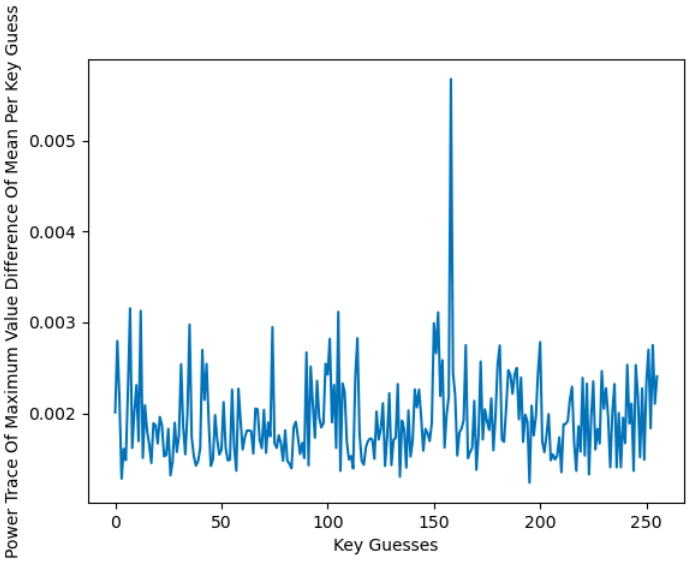

Figure 26: DPA for 0^{th} subkey guess performed for 1000 traces.

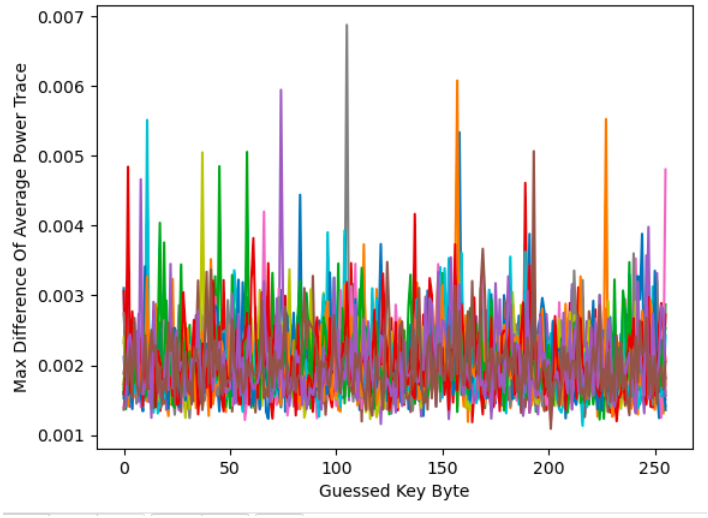

Figure 27: DPA for all subkey guesses performed for 1000 traces.

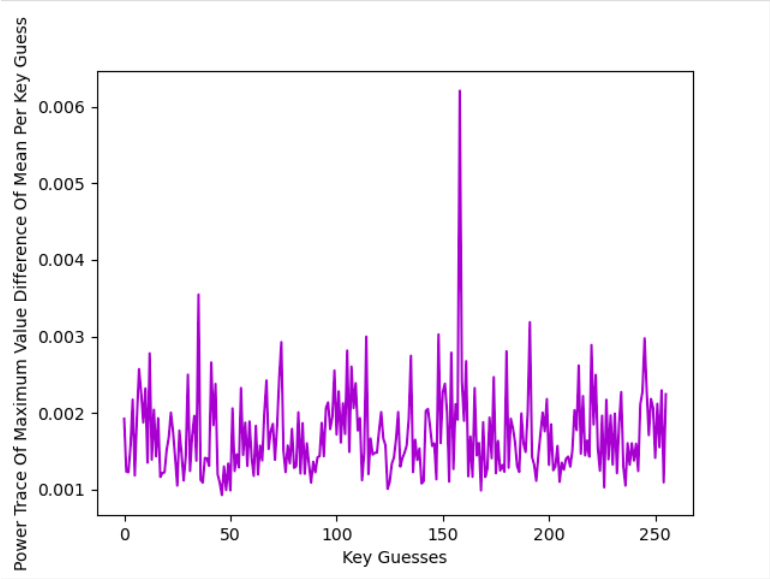

Figure 28: DPA for 0^{th} subkey guess performed for 2000 traces.

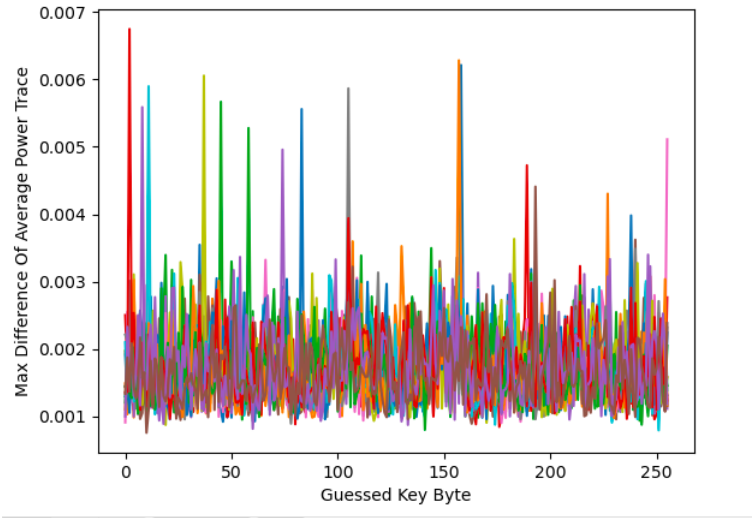

Figure 29: DPA on all subkey guesses performed for 2000 traces.

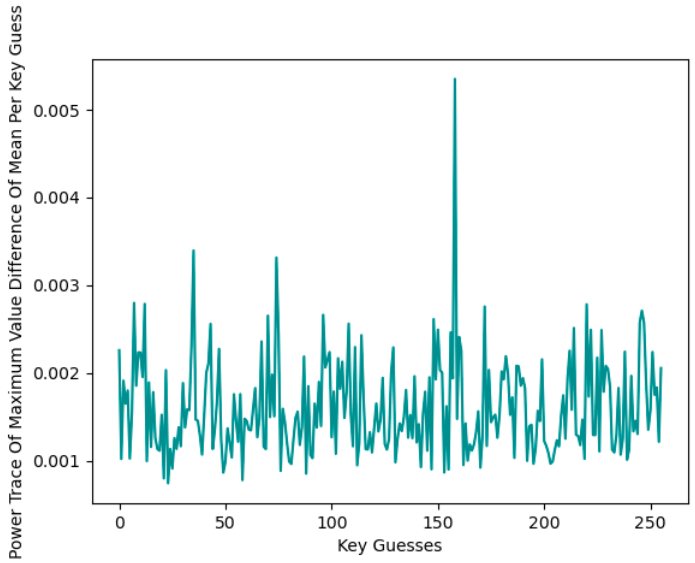

Figure 30: DPA on all subkey guesses performed for 3000 traces.

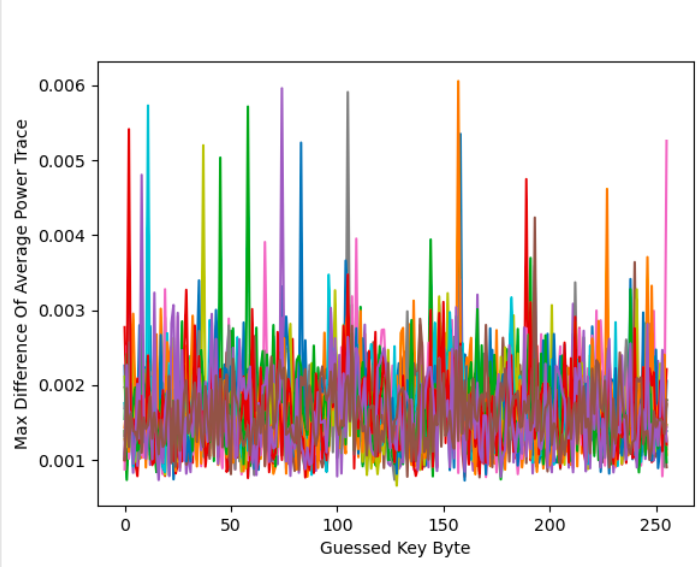

Figure 31: DPA on all subkey guesses performed for 3000 traces.

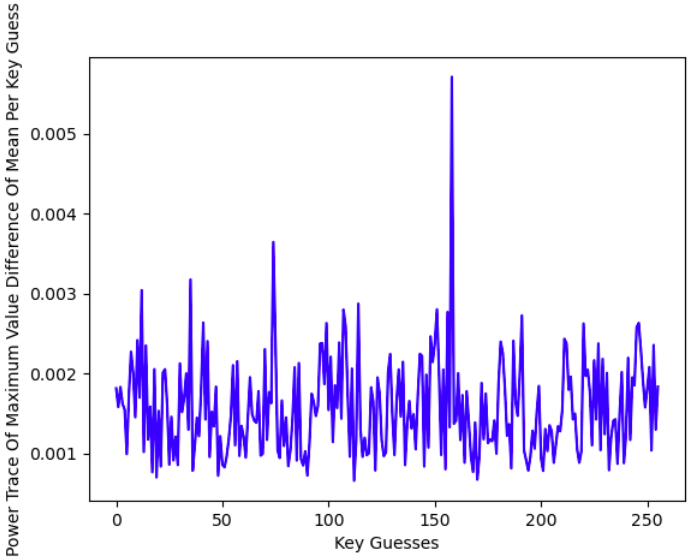

Figure 32: DPA for 0^{th} subkey guess performed for 5000 traces.

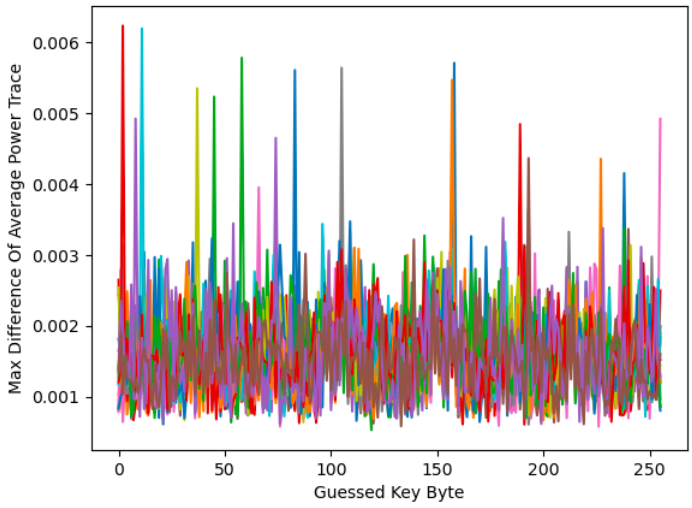

Figure 33: DPA on all subkey guesses performed for 5000 traces.

3.6.4 Conclusion

In our experiments, we were able to reveal the key by attacking the last bit of the hypothetical leakage value for all key guesses. The spikes in the power trace plot align with the actual key.

3.7 Correlation Power Attacks on Firmware Implementation

In the previous section, we implemented differential power attacks to reveal the AES key. We were successfully able to implement the exploit the intermediate value of the AES to fetch the key stored in the device. However, this takes a large number of traces to break the AES. We will now look at a more effective type of side channel power attack known as Correlation Power Attack.

In correlation power attack, we look at the Hamming weight and power relation of the Sbox intermediate value and try to reveal the key instead of comparing the difference of the mean of the power trace for different key guesses. However, first let let us try to determine the relationship between the Hamming weight and power.

3.7.1 Hamming Weight Vs Power Trace For AES Sbox

We investigate the effect of Hamming weight on the power consumption calculated while running the AES algorithm. We group the power traces based on the Hamming weight of the internal value calculated using the random plain text, fixed key, and Sbox. Subsequently, we calculate the average trace for each Hamming weight class and the average of all power traces and subtract the two values. We plot the results while varying the Hamming weight values. We observe from the plot that the points where the color separates are actually the points where the Sbox implementation is happening in the trace since we are observing the power traces for these intermediate AES values.

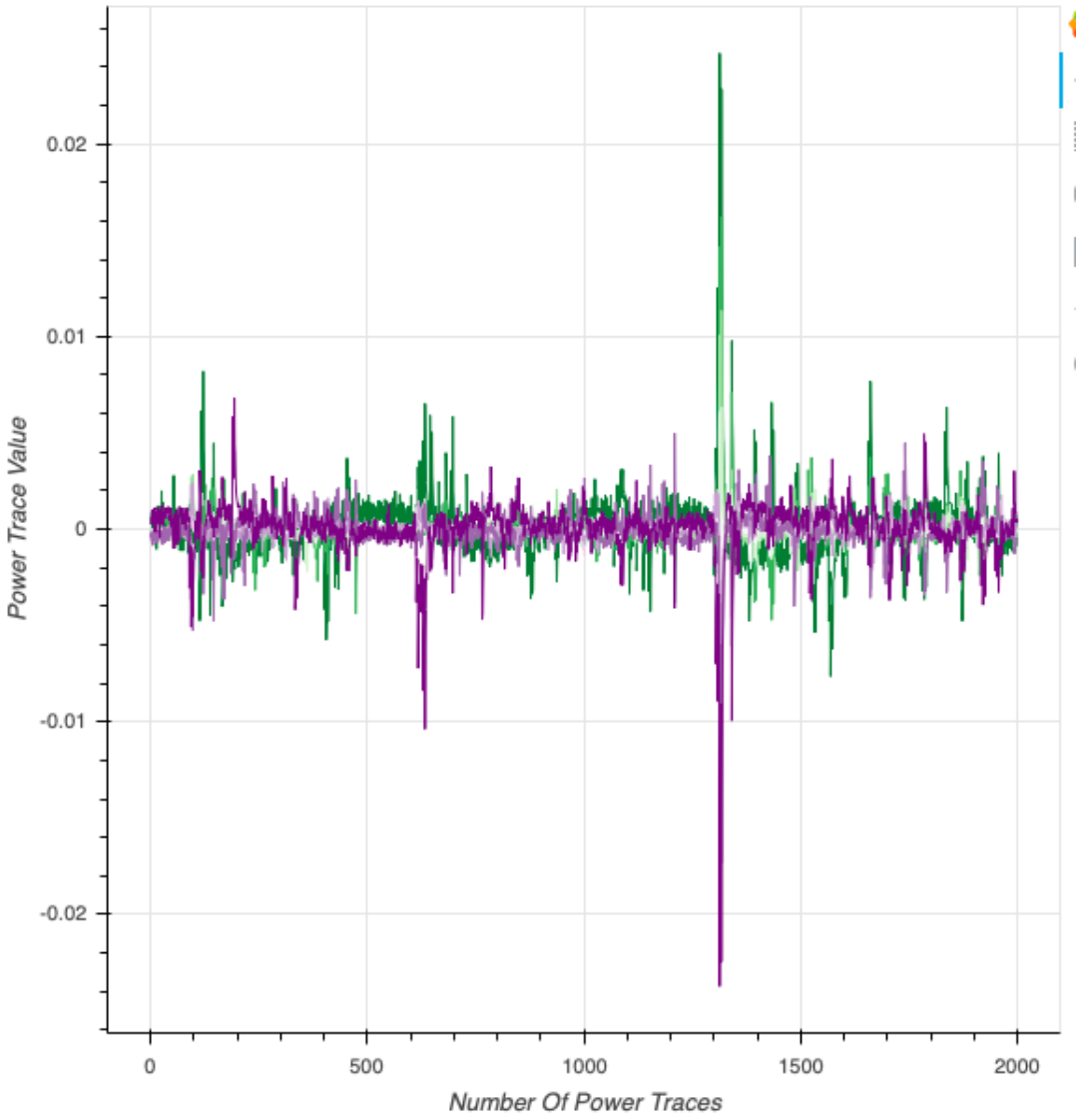

Figure 34: Sbox Output Detection Using Power Trace.

From the above diagram, it can be seen that the Hamming weight has a linear relationship with voltage except for Hamming weight 4 since it is the value that occurs the most frequently and thus the average trace is comparatively less compared to other values. Therefore, we can safely assume that those intermediate values which have a linear relationship with the power trace might be closest to the actual key. We can use this for Correlation Power Attacks.

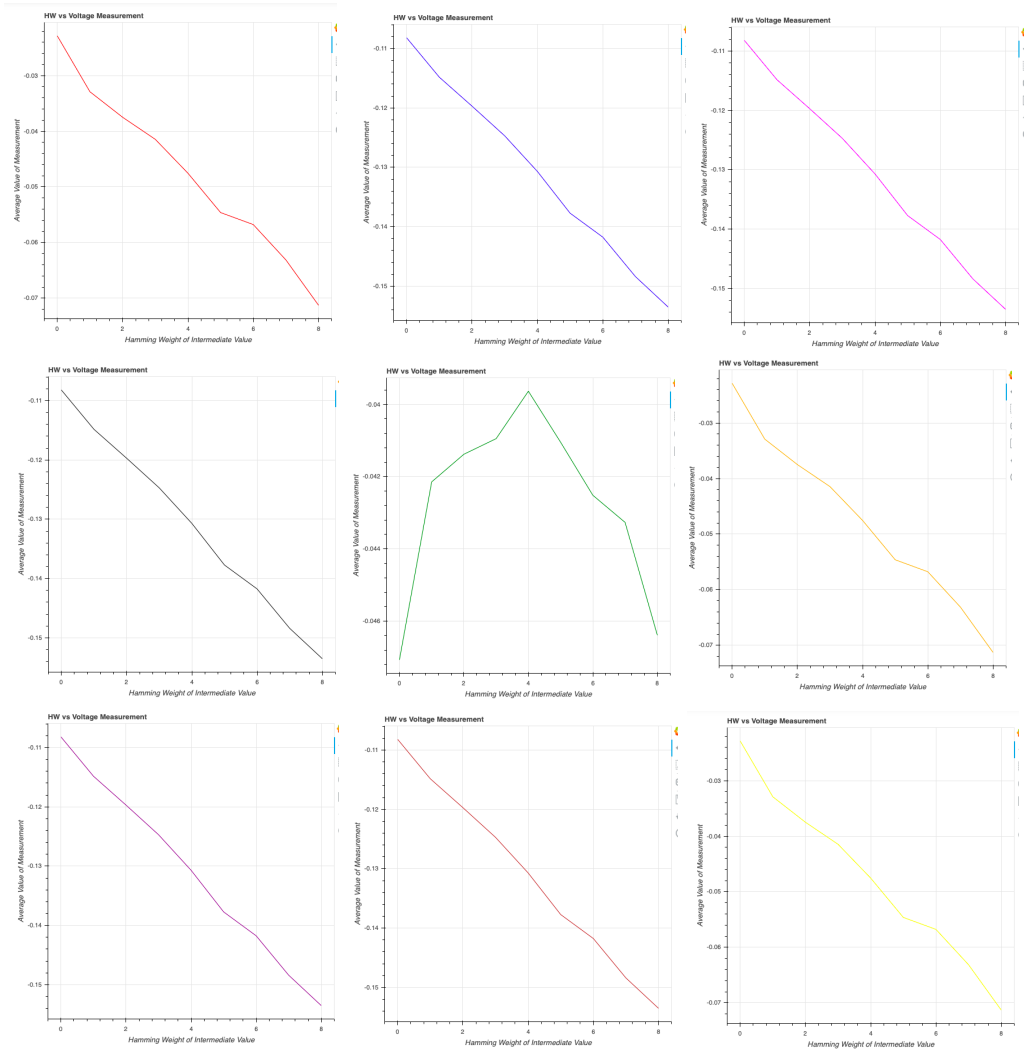

Figure 35: Hamming Weight vs Voltage Relationship For All Possible Hamming Weights.

3.7.2 Experiment

Algorithm 6: Correlation Power Analysis.

```
Result: Correlation Matrix Calculated
Input: powertrace[Samples][Points];
Output: correlationmatrix[Key Guess][Points];
meantracearray = mean(tracearray);
std dev trace=standard deviation(tracearray, meantracearray);
for key guess in range(0,255) do
    for text in text array do
        hammingweightarray = transpose(hamming weight(sbox[keyguess ⊕
          plaintext])) ;
        hammingweightarraymean = mean(hammingweightarray) ;
        std dev hamming weight = standarddeviation(hammingweightarray,
          hammingweight arraymean);
        correlation=covariance(tracearray, tracemean, hammingweightarray,
          hamming weightarraymean);
        correlationoutput = correlation/(stddevtrace*stdhammingweight);
        maximumcorrelationoutput[keyguess] =
          maximum(absolute(correlationoutput));
    end
end
```

The actual power is stored in the trace_array which we observe and compute the average for each plain text value. Instead of computing the hypothetical leakage using only the internal value, we calculate it by computing the Hamming weight for all plain text values and using the average to compute the correlation between the actual power trace & its average and the corresponding Hamming weight and Hamming weight average. This is based on the fact that the power trace is directly proportional to the Hamming weight of Sbox internal value. We performed the correlation power attack while varying the number of traces. The experiment was performed for 25, 30, 40 & 50 trace values and plotted the correlation matrix vs the number of key guesses for each key byte and later for all key bytes. We compared the guessed key with the actual key to check the effectiveness of the attack.

3.7.3 Results

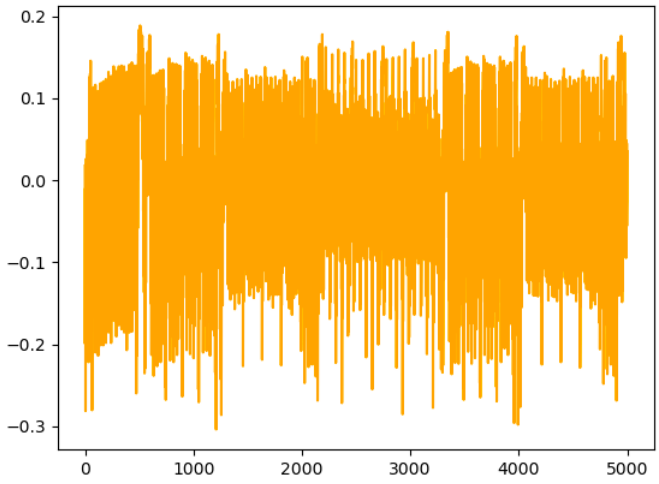

Figure 36: Power Trace Collected For 1^{st} Plaintext When N=25.

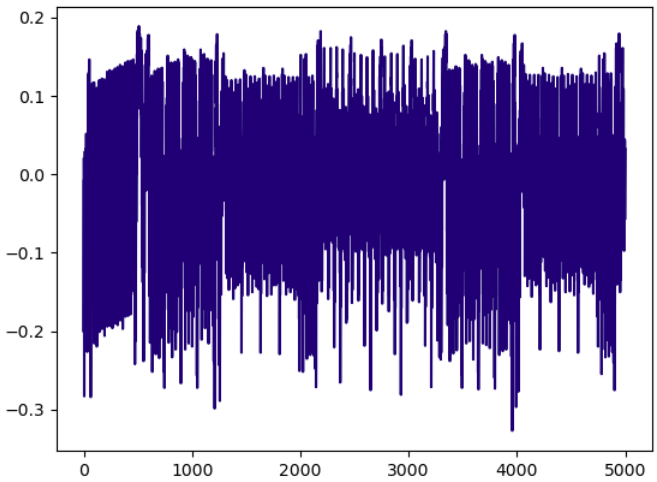

Figure 37: Power Trace Collected For 1^{st} Plaintext When N=30.

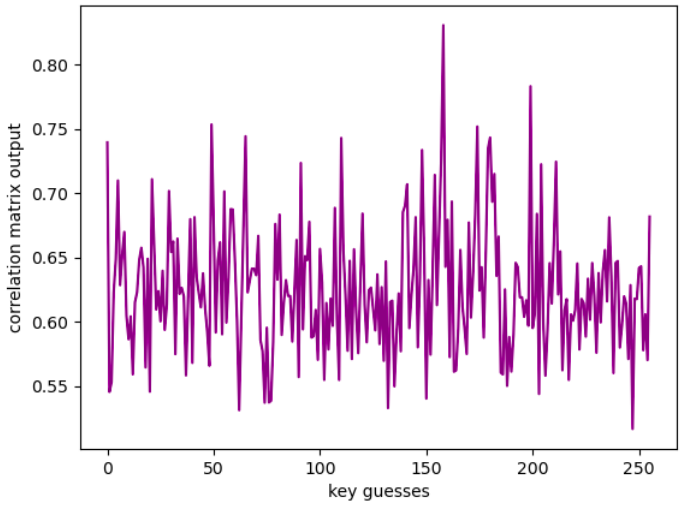

Figure 38: Correlation Power Attack For 25 Traces on 0^{th} Subkey Byte.

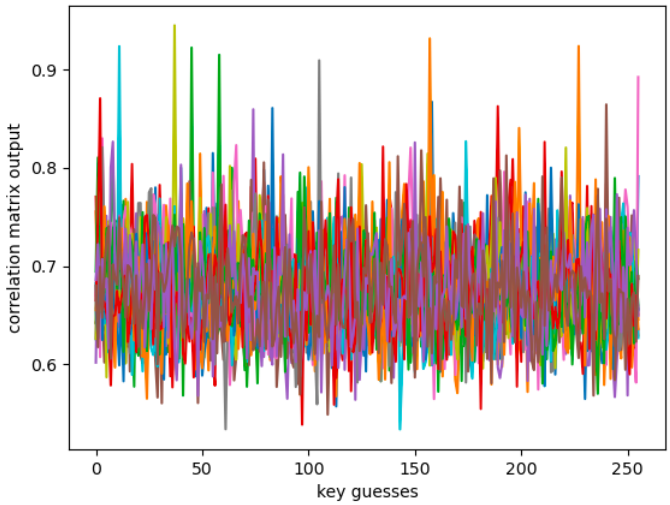

Figure 39: Correlation Power Attack For 25 Traces On All Subkey Bytes.

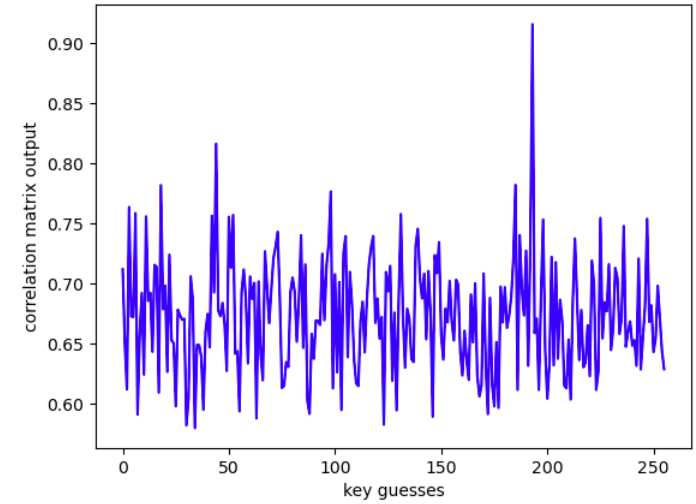

Figure 40: Correlation Power Attack For 30 Traces on 0^{th} Subkey Byte.

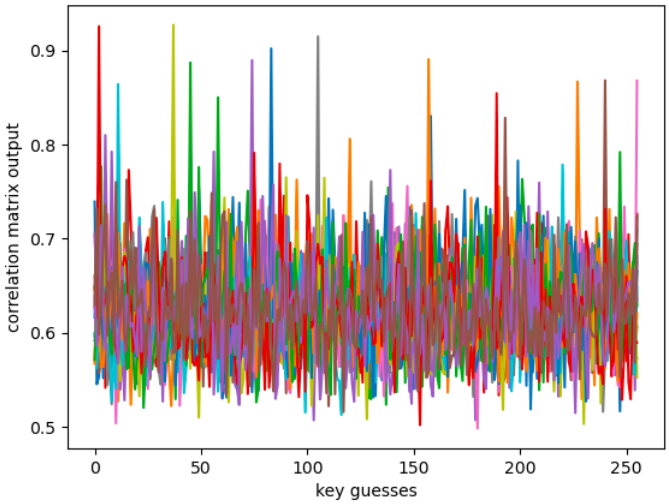

Figure 41: Correlation Power Attack For 30 Traces On All Subkey Bytes.

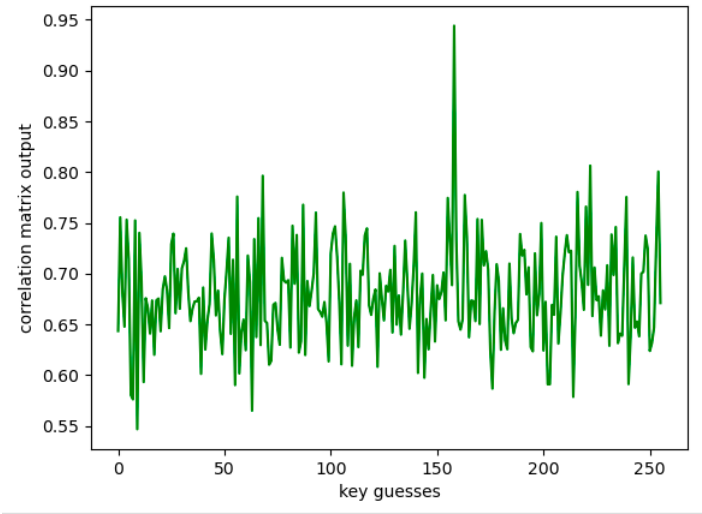

Figure 42: Correlation Power Attack For 40 Traces On 0^{th} Subkey Byte.

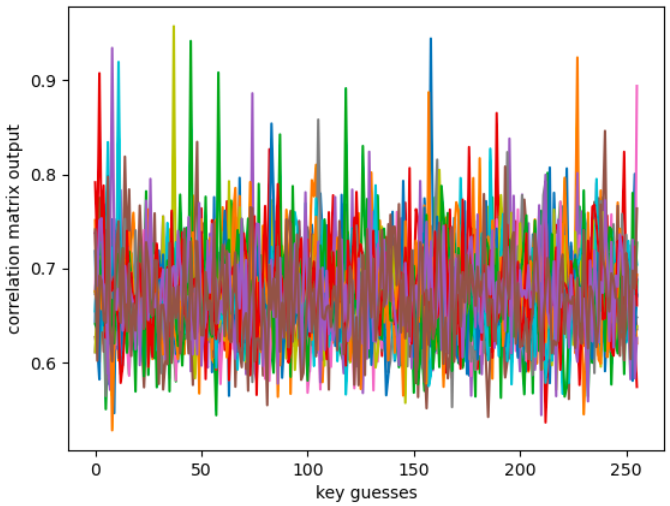

Figure 43: Correlation Power Attack For 40 Traces On All Subkey Byte.

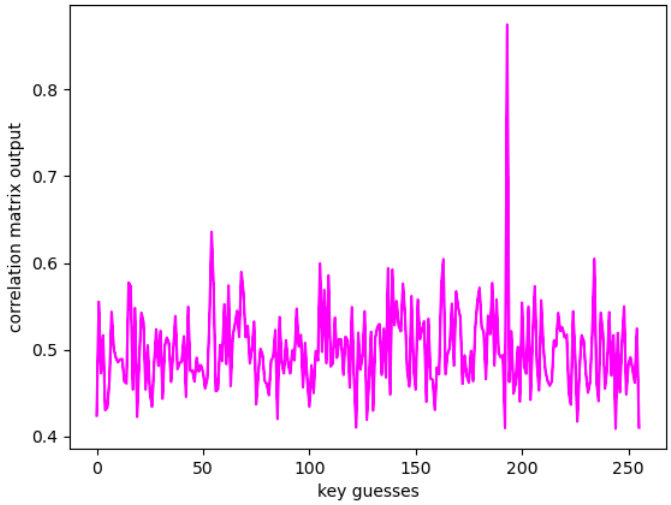

Figure 44: Correlation Power Attack For 50 Traces On 0^{th} Subkey Byte.

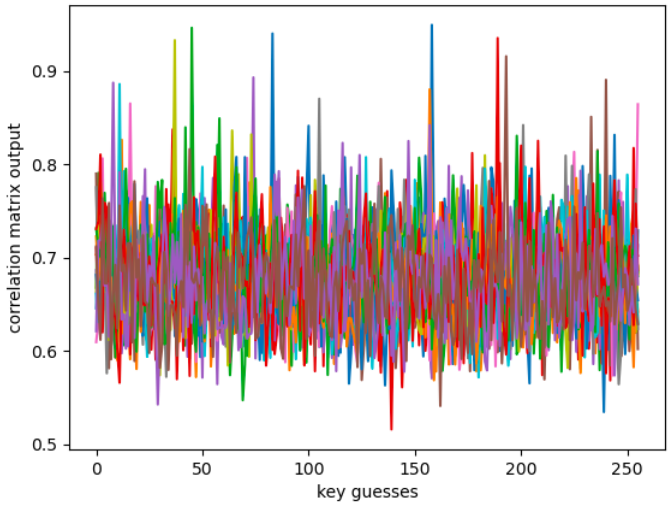

Figure 45: Correlation Power Attack For 50 Traces On All Subkey Byte.

3.7.4 Conclusion

From the results, we can conclude that Correlation Power Attacks require much lower amount of power traces to break the AES implementation. For instance, for 25 traces we were able to guess a lot of subkeys correctly, but as soon as we increased the number of traces to 30, the guessed key was equal to the actual key. Moreover, the peak of the guessed key became much more prominent and distinguishable as we increased the number of traces from 25 to 50. Thus, correlation power attack is a more efficient and stronger attack than Differential Power Attack. Moreover, the power trace collected when we run for any number of traces remains consistent.

3.8 Voltage Glitching

Inserting brief glitches into the power line of an embedded device can result in skipped instructions and corrupted results. Since Vcc pin on the microcontroller is more accessible. This concept can be used to insert a fault in your device resulting in the processor skipping the authentication part entirely. We will be investigating the effect of inserting glitches on the power trace obtained from the STM 32 and 8-bit microcontrollers.

3.8.1 Glitch Hardware

The ChipWhisperer Glitch system uses the same synchronous methodology as its Side Channel Analysis (SCA) capture. A system clock (which can come from either the ChipWhisperer or the Device Under Test (DUT)) is used to generate the glitches. These glitches are then inserted back into the clock, although it is possible to use the glitches alone for other purposes (i.e., for voltage glitching, EM glitching).The generation of glitches is done with two variable phase shift modules, configured as follows:

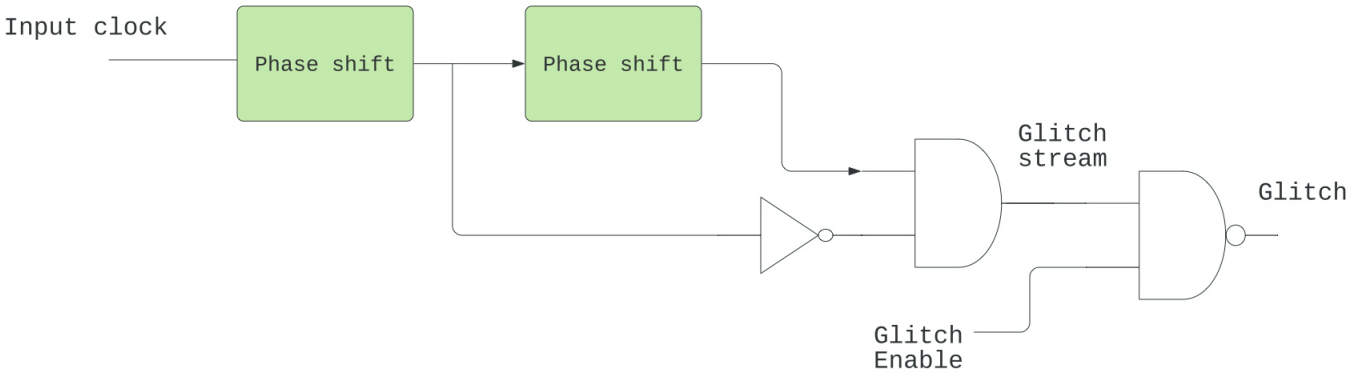

Figure 46: Setup For Obtaining Glitch.

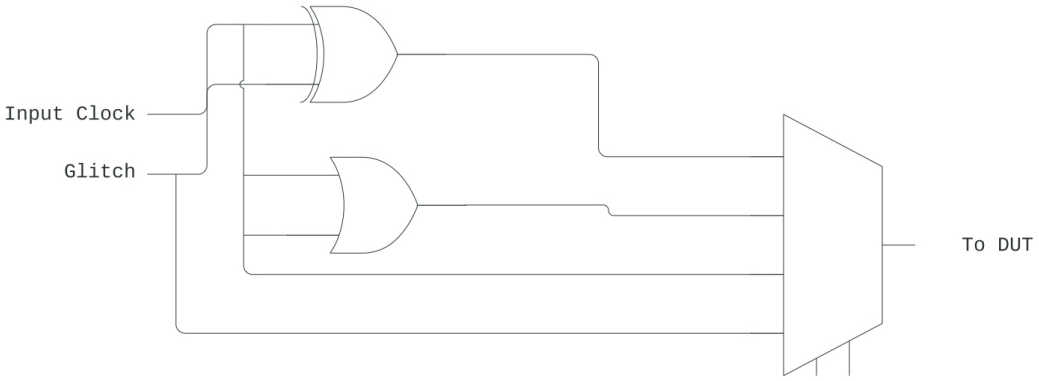

Figure 47: Glitch to DUT.

Voltage Glitch Setup

The chipwhisperer routs the glitch output to either *glitch_lp* or *glitch_hp*. *glitch_lp* is the glitch output which is enabled when the low-power crowbar MOSFET is enabled and *glitch_hp* is when high-power crowbar MOSFET is enabled. Based on which setting is enabled, that particular MOSFET shorts the power rail to the ground when the glitch module's output is active. There are two modes which we can use to control the width of the glitch.

- *glitch_only*: Insert a glitch only for a part of the clock cycle. You can control the width and the offset value of the glitch. We will be using *repeat*, *width*, ext_offset or external offset. In this mode we insert an asynchronous glitch to the clock, this can cause the device to malfunction sometimes.

- *enable_only*: Insert a glitch for the entire duration of the clock. These settings are synchronous with the clock, so there is less chance of setup and hold time violations in the design. However, glitching for the entire clock cycle can be too strong for most devices.

Between the two modes, we performed my experiment in *glitch_only* mode since we were performing the experiment on a low power microcontroller.

Table 3: Operating Conditions Of ADC For Voltage Glitching.

ADC Parameters	**Values**
ADC clock source	clkgen
Glitch Width	10.15625
Offset	10.15625
Glitch Mode	*glitch_only*
External Offset Value	0
Repeat	0
ADC gain	24.8359375 dB
Serial Baud Rate	125047.48982360923
ADC frequency counter source	external clock
ADC Frequency	24MHz
ADC clkgen multiplier	2
ADC clkgen divider	26
ADC clkgen frequency	7384615.384615385
ADC clkgen locked	True

3.8.2 Experiment

Unlike clock glitching, we cannot oscillate the V_{cc} value multiple times as it may damage the device. Therefore, we will increase the glitch width until we see a significant number of crashes. We have a repeat parameter which resets the V_{cc} value for the next iteration of glitching so that the scope does not stop after V_{cc} is unable to go back to its original value.

Glitch Controller

We use a software construct called glitch controller to control different aspects of our glitch. It has the following parameters:

- **Width:** Glitch width iterating from minimum to maximum range during a particular offset value. We set the range from 5 to 48.
- **Offset:** Glitch offset iterates over the set range -40 to 40.
- **External Offset:** Setting different offset values for different iterations of the glitch.
- **Range:** We set the range of width and offset & external offset using this parameter. The default range we set was from 5 to 20.
- **Reset Count:** To count the number of times the system crashes during glitching, we use this parameter. We matched the return value we received during a simple serial glitch of the target. If the value is false, we add 1 to the reset count.
- **Success Count:** The number of successes we obtained while glitching is recorded using the success parameter.
- **Normal Count:** During glitching, there are many instances where the glitching is not successful. This parameter stores the count during the experiment.

We collect the power traces while running the glitched simple serial communication in which we set the trigger to high for 2500 iterations and then to trigger low when the count reaches 2500 iterations, it resets. After the trigger is set to low, we read 4 bytes of data. If the glitch is successful, then the data being read is leaked by the glitch.

3.8.3 Results

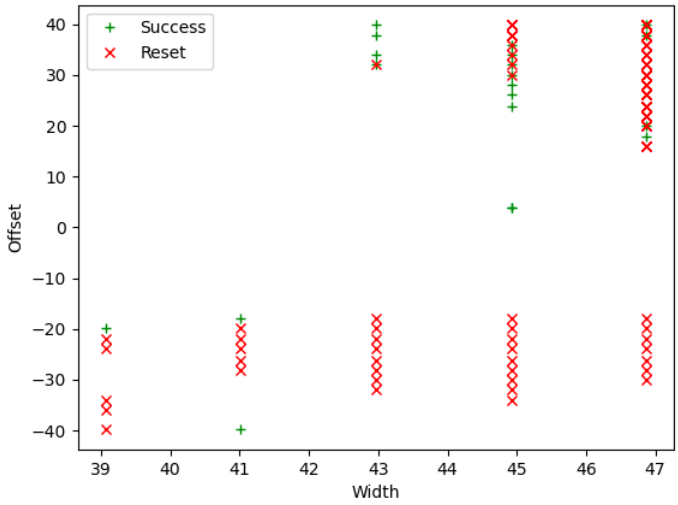

Figure 48: Number of Successes & Reset For Glitch Width -40 to 40.

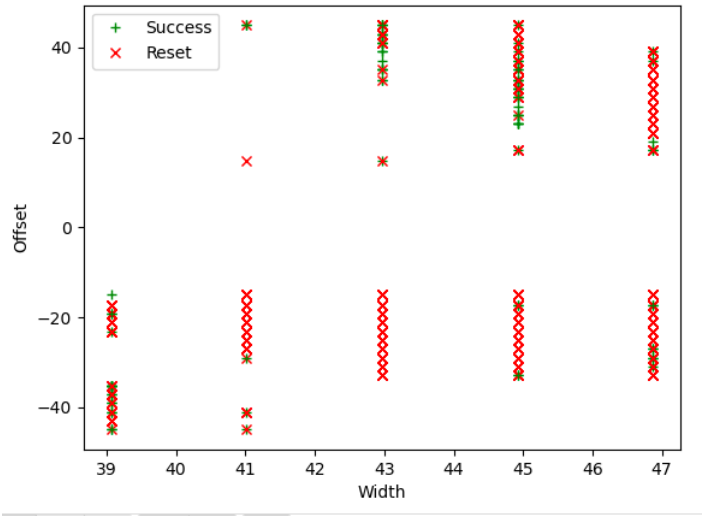

Figure 49: Successes & Reset For Glitch Width -45 to 45.

3.8.4 Conclusion

No successful glitching or reset occurs for a glitch width lesser than 39 and the text is written normally into the microcontroller. Moreover, the number of successes and resets are irregular since we are glitching asynchronously. We observed that a successful glitch pattern even for the same conditions does not exist. This is difficult

in the case of an attack where we require consistent results.

3.9 Clock Glitching

Digital hardware requires a consistent synchronous clock to function optimally and predictably. If the clock becomes erratic, then the performance of the hardware becomes unpredictable as the instruction execution is controlled by the clock edge. We can manipulate these clocks to cause unintended behavior just like in the case of voltage glitching. Lets take the example of a typical microcontroller in which the instructions are fetched from the flash memory using pipelining. The instructions are fetched based on the clock edge of the synchronous system clock which is provided to the microcontroller chip. However, if the clock becomes asynchronous, then some of the instructions that are arriving at the instruction register might get skipped. Assume the instructions that are skipped are used by the microcontroller to authenticate for a password. In that case, the device becomes vulnerable since the security measures used for safeguarding have been compromised.

3.9.1 Hardware Setup

The phase shift blocks use the Digital Clock Manager (DCM) blocks within the FPGA. These blocks have limited support for run-time configuration of parameters such as phase delay and frequency generation, and for maximum performance the configuration must be fixed at design time. The Xilinx-provided run-time adjustment can shift the phase by about +/- 5ns in 30ps increments (exact values vary with operating conditions). For most operating conditions, this is insufficient - if attacking a target at 7.37MHz, the clock cycle would have a period of 136nS. To provide a larger adjustment range, an advanced FPGA feature called Partial Reconfiguration (PR) is used. The PR system requires special partial bitstreams which contain modifications

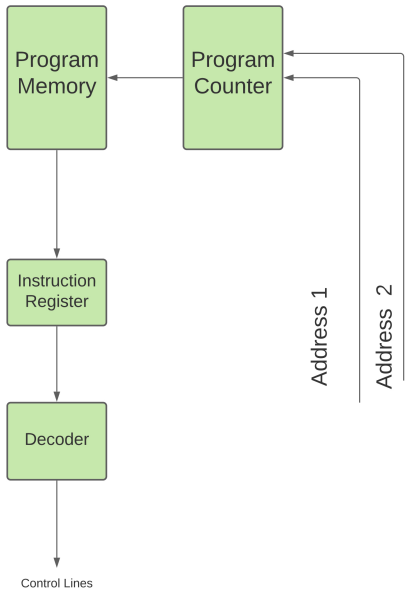

Figure 50: Microcontroller Architecture.

to the FPGA bitstream. These are stored as two files inside a "firmware" zip which contains both the FPGA bitstream along with a file called glitchwidth.p and a file called glitchoffset.p. If a lone bitstream is being loaded into the FPGA (i.e., not from the zip file), the partial reconfiguration system is disabled, as loading incorrect partial reconfiguration files could damage the FPGA. This damage is mostly theoretical, more likely the FPGA will fail to function correctly. If during following this tutorial you find the FPGA appears to stop responding (i.e., certain features no longer work correctly), it could be the part reconfiguration data is incorrect.

3.9.2 Glitch Module

Just like voltage glitching, we use the glitch controller to control the amount of glitch. Unlike the voltage glitch, we can oscillate the clock and glitch multiple times as we do not need to maintain a constant supply of the signal for the basic functionality

of the device. The parameters we will be controlling are:

- **Clock Source:** The clock signal that the glitch DCM is using as input. Can be set to "target" or "clkgen" In this case, we will be providing the clock to the target, so we will want this set to "clkgen"

- **Offset:**Where in the output clock to place the glitch. Can be in the range [-50, 50]. Often, we will want to try many offsets when trying to glitch a target.

- **Width:** How wide to make the glitch. can be in the range [-50, 50], although there is no reason to use widths ¡ 0. Wider glitches more easily cause glitches, but are also more likely to crash the target, meaning we will often want to try a range of widths when attacking a target.

- **Output:**The output produced by the glitch module. For clock glitching, *clock_xor* is often the most useful option.

- External Offset: The number of clock cycles after the trigger to put the glitch.

- **Repeat:** The number of clock cycles to repeat the glitch for. Higher values increase the number of instructions that can be glitched, but often increase the risk of crashing the target.

- **Trigger Source:** How to trigger the glitch. In our lab, we want to automatically trigger the glitch in the trigger pin only after arming the ChipWhisperer.

3.9.3 Experiment

We perform clock glitching for different width, offset, and external offset ranges. We start the experiment with a glitch width in the range of -20 to 20 and offset in the range of -40 to 40 and step size of 8 to 4 and repeat 2. We repeated the experiment multiple times to observe the consistency of the glitch.

Later we change the settings to offset -30 to 30, -35 to 35, -25 to 35 width from -15 to 15, -25 to 25 and observe the results. We cannot have the range of offsets above 50 as the clock glitching does not work beyond this range. We plot the result at the end.

3.9.4 Results

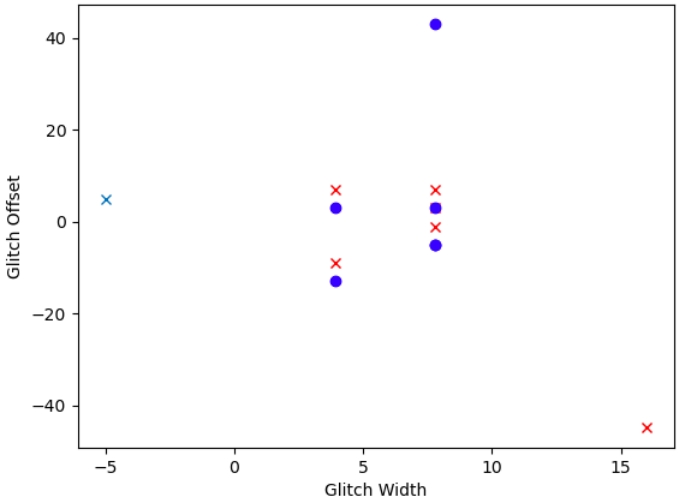

Figure 51: Clock Glitching : Width -20 to 20, Offset -40 to 40 First Attempt.

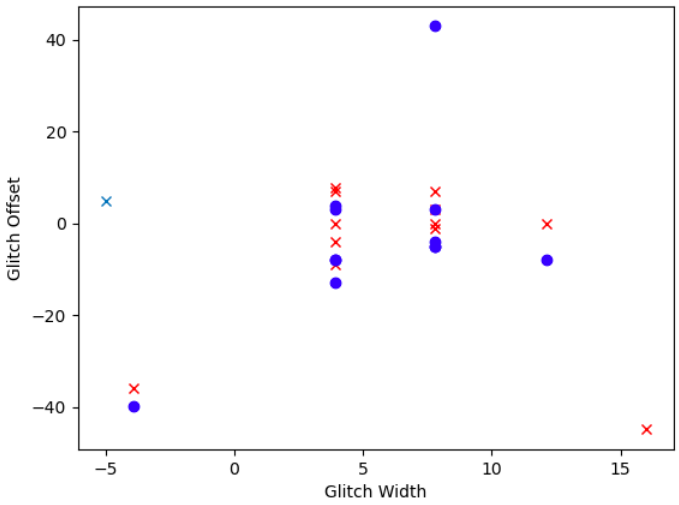

Figure 52: Clock Glitching : Width -20 to 20, Offset -40 to 40 Second Attempt.

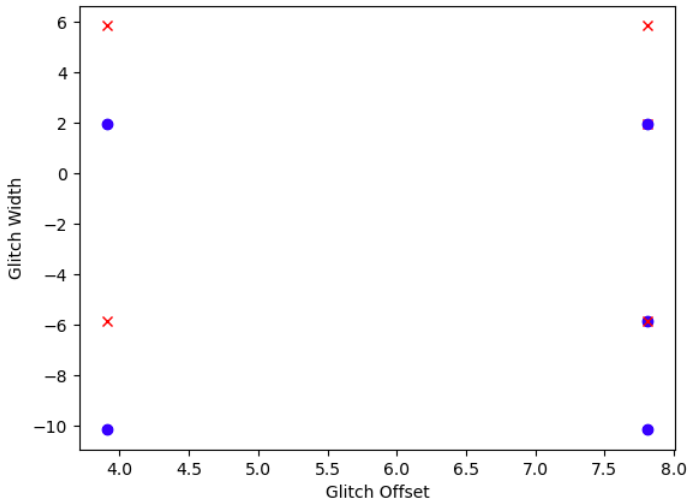

Figure 53: Clock Glitching : Width -8 to 4, Offset -10 to 6 First Attempt.

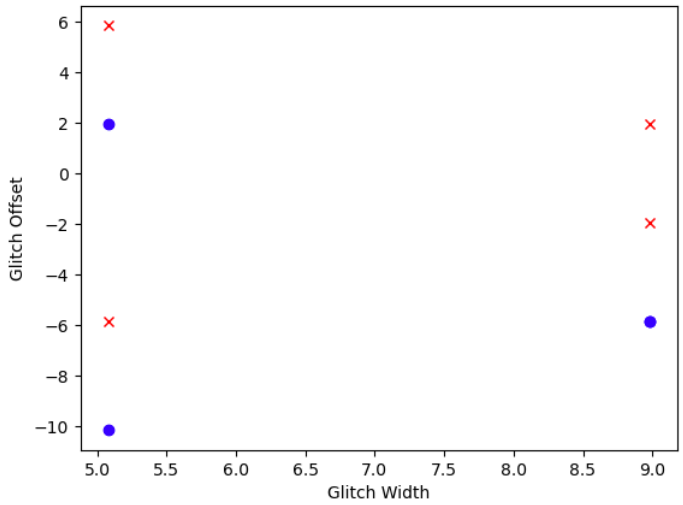

Figure 54: Clock Glitching : Width -15 to 15, Offset -30 to 30 First Run.

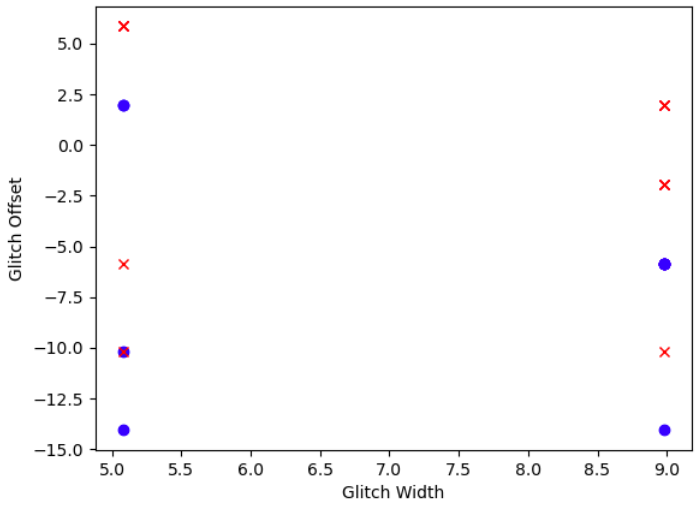

Figure 55: Clock Glitching : Width -15 to 15, Offset -30 to 30 Second Attempt.

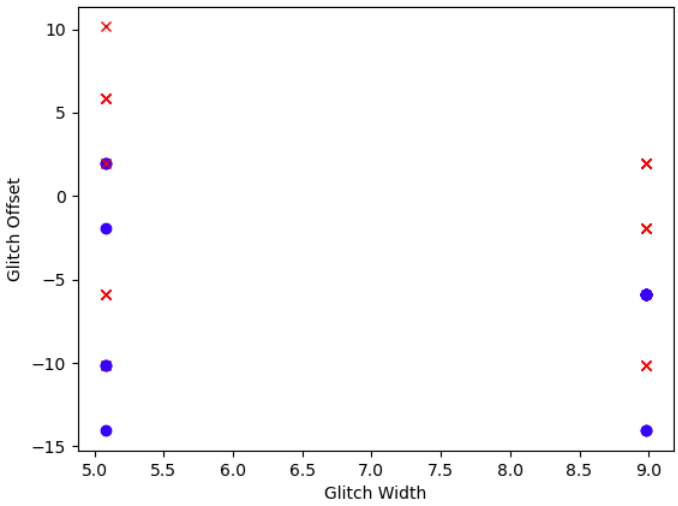

Figure 56: Clock Glitching : Width -15 to 15, Offset -30 to 30 Third Attempt.

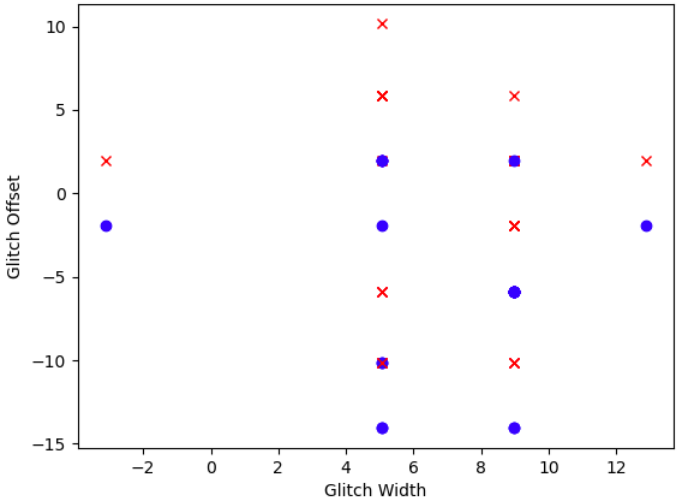

Figure 57: Clock Glitching : Width -15 to 15, Offset -30 to 30 Fourth Attempt.

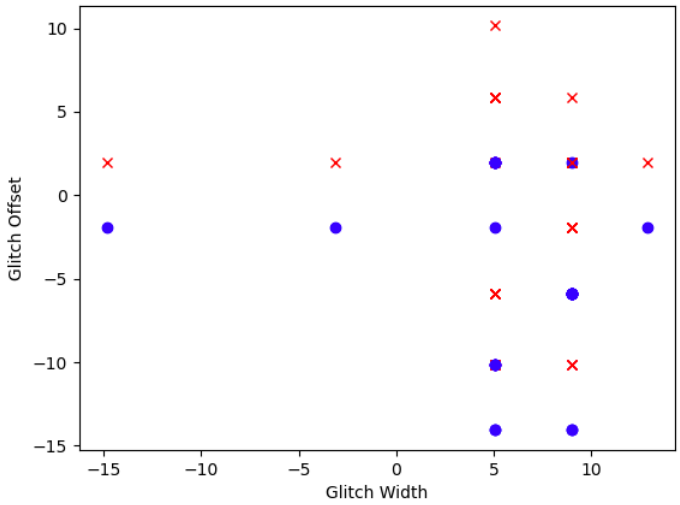

Figure 58: Clock Glitching : Width -15 to 15, Offset -30 to 30 Fifth Attempt.

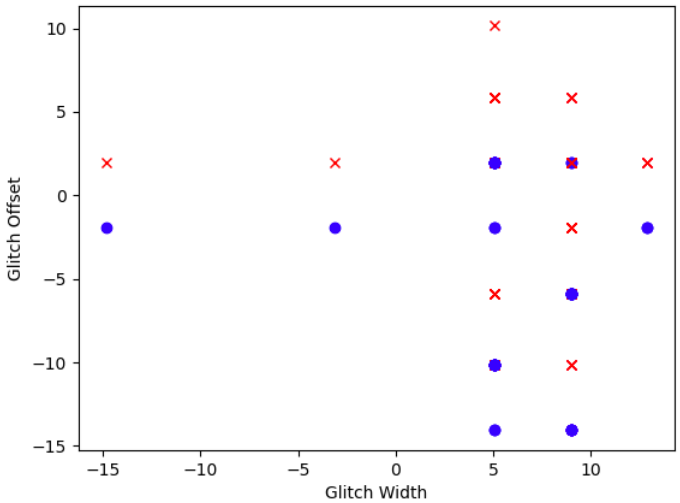

Figure 59: Clock Glitching : Width -15 to 15, Offset -30 to 30 Tenth Attempt.

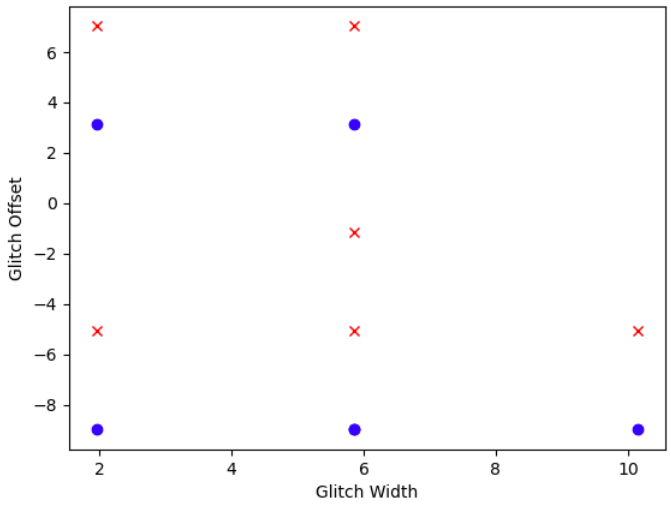

Figure 60: Clock Glitching : Width -10 to 10, Offset -25 to 25 First Attempt.

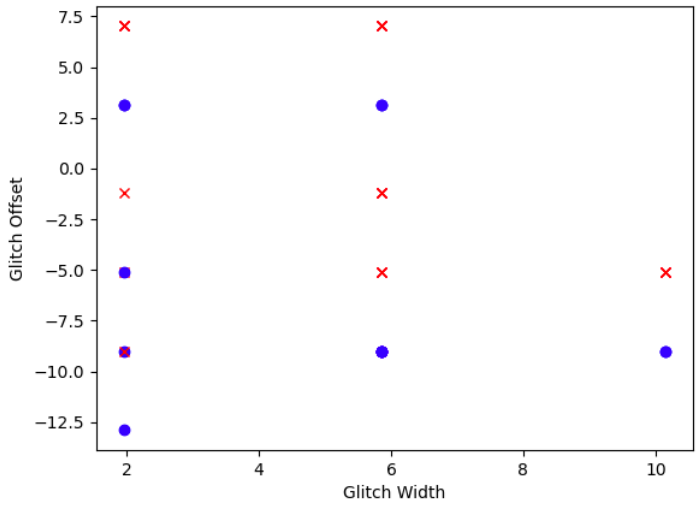

Figure 61: Clock Glitching : Width -10 to 10, Offset -25 to 25 Second Attempt.

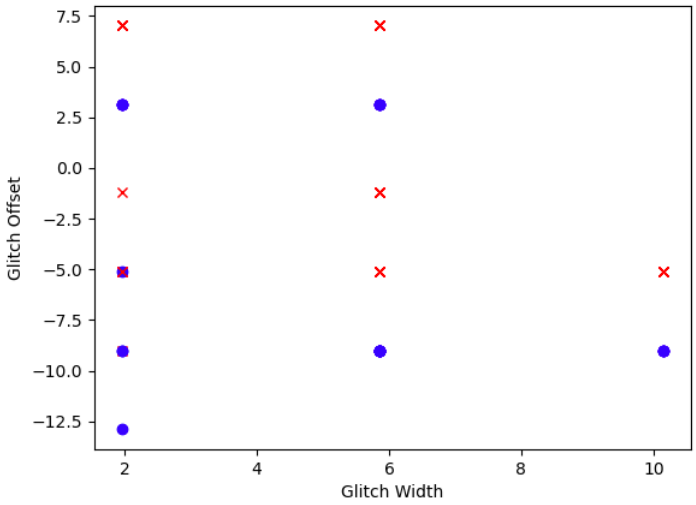

Figure 62: Clock Glitching : Width -10 to 10, Offset -25 to 25 Third Attempt.

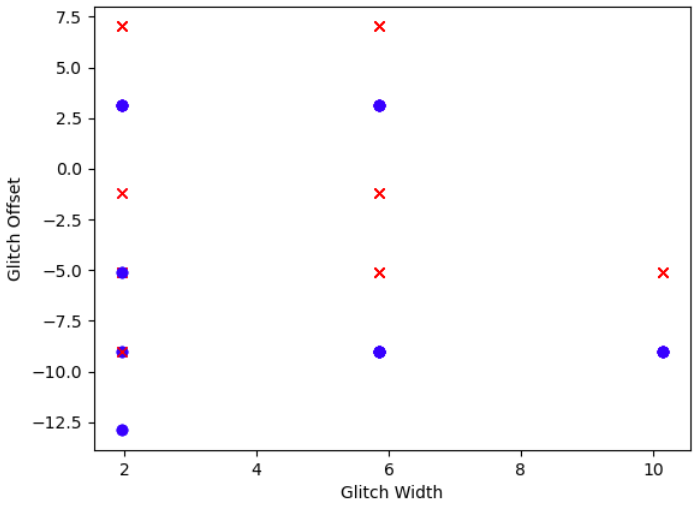

Figure 63: Clock Glitching : Width -10 to 10, Offset -25 to 25 Fourth Attempt.

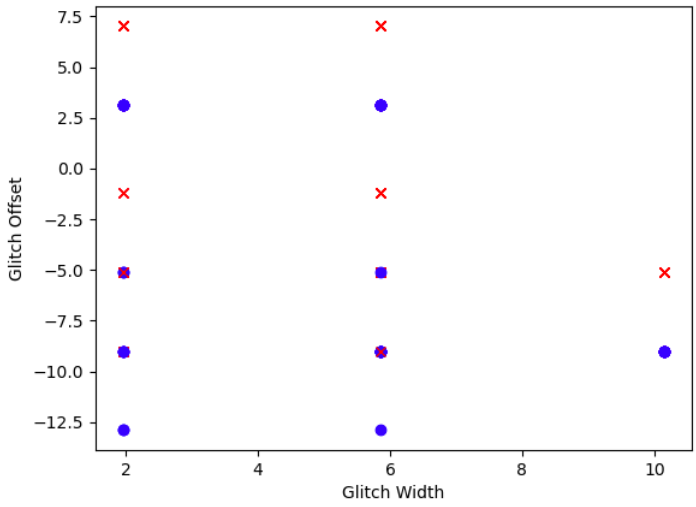

Figure 64: Clock Glitching : Width -10 to 10, Offset -25 to 25 Fifth Attempt.

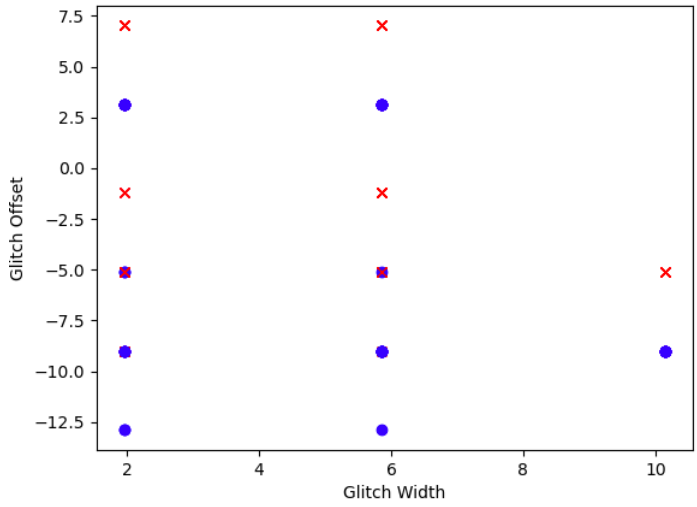

Figure 65: Clock Glitching : Width -10 to 10, Offset -25 to 25 Sixth Attempt.

3.9.5 Conclusion

From our experiment, we observed that clock glitching shows consistent results over multiple runs as we decrease the range of width and offset. The number of successful glitches are greater than the normal runs compared to those for voltage glitching.

3.9.6 Voltage Vs Clock Glitching

Voltage glitching has some obvious benefits over clock glitching, such as working for a wider variety of targets, but its downsides are less obvious. One of the biggest is how much it depends on the actual glitch circuit itself. With clock glitching, it its relatively easy to insert a glitch - there is nothing external trying to keep the clock at a certain voltage level. This is very different from a target's power pin. When we try to drop the power pin to the ground, there is a lot of stuff fighting us to keep the power pin at the correct voltage, such as decoupling capacitors, bulk supply capacitors, and the voltage regulator supplying the voltage. This means when we make small changes to the glitch circuit, the glitch settings and even our ability to insert a glitch at all completely change.

While we first thought that it is better to go for as sharp glitches as possible, this often will not result in a high glitch success rate. If you are unable to find any working glitches on your current setup, it might be worth changing you a hardware setup bit. For example, on the ChipWhisperer Lite part, you can desolder SJ5 and solder header pins to JP6. Even just connecting these pins with a jumper will have a different glitch behaviour than with a soldered SJ5. You can refer to the training slides for more information about finding good glitch settings, as well as more on the theory side of voltage glitching.

3.10 Differential Fault Attacks

The principle is to repeat the same AES operation over and over and to glitch its intermediate operations to get an output cryptographically incorrect. In this attack, we only target a single byte of the last or second last mixcolumns of AES. Using the DFA, we will are able to recover the last round key of the AES with various computations that may be quite intensive. We attack a single byte of the target output while keeping the plain text and key constant. AES-128 is made of 10 rounds, the last one is missing the Mix Column operation, the only operation which brings diffusion, i.e., an operation which makes a single byte of one round state affecting multiple bytes in the next round, 4 bytes to be exact. We inject the fault between two MixColumn operations and propagate 4 of the 16 output bytes will be wrong. We can inject our faults wherever we want and simply observe the output and look for a 4-byte fault with one of the 4 possible patterns. We then recover the entire key by comparing the correct output and the faulty outputs, and because the AES keyschedule is inverted, we can compute it backwards and recover the first round key, which is by definition equal to the AES-128 key. We capture the last trace keeping the ADC samples = 10000. Keeping the plain text key and cipher text constant. The value of plain text does not matter, it just needs to be constant. The text output received from a single trace is referred to as a gold cipher and it has the following data sets stored in it.

- **Plaintext:** ec4667567a55a56c325cf9e1adfe6dab
- **Key:** 9ee32dbd4af0ff69250b539d3a0208c1
- **Ciphertext:** 77c0f9a76568263eccecd86347aef416

We use glitch controller to glitch the the 9^{th} & 10^{th} rounds of AES implementation. We need to keep on changing the glitch parameters to get successful glitches.

We carry out our first glitch using the following parameters.

- **glitch width:** -10.15625
- **glitch offset:** -39.84
- **external offset value:** 5400

3.10.1 Glitch Campaign

We use a more advanced version of the glitch controller called as glitch campaign to gather results and output while repeatedly glitching the mix columns. For each iteration of glitch values that we choose from the glitch controller, we capture the scope trace and text output. If the output byte is same as the corresponding golden cipher byte, that means no glitch was performed and we categorise the corresponding text and scope value under normal. For the glitched plaintext, we note the corresponding glitched key and note the corresponding column number, width, external offset, and offset values. After obtaining the faulty ciphertexts, we use the phoenixAES library for performing differential fault analysis for the last column of AES. In the final step, we revert the entire AES key schedule and reveal the actual key successfully.

If unsuccessful, we run the glitch campaign function again with a different set of glitch controller values until we are able to reveal the key.

3.10.2 Results

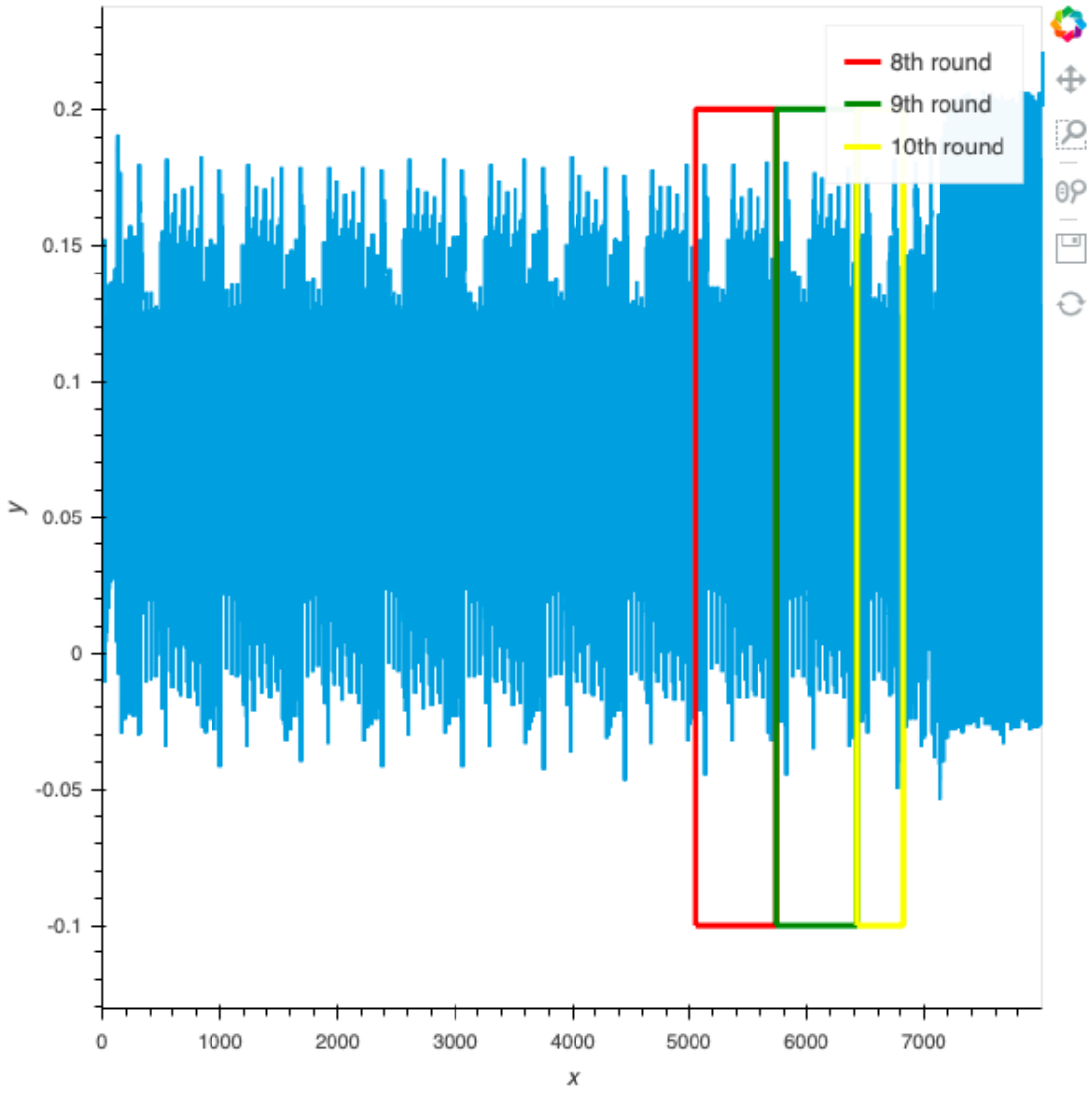

Figure 66: Power Trace Of AES Highlighting 8^{th}, 9^{th} & 10^{th} Rounds.

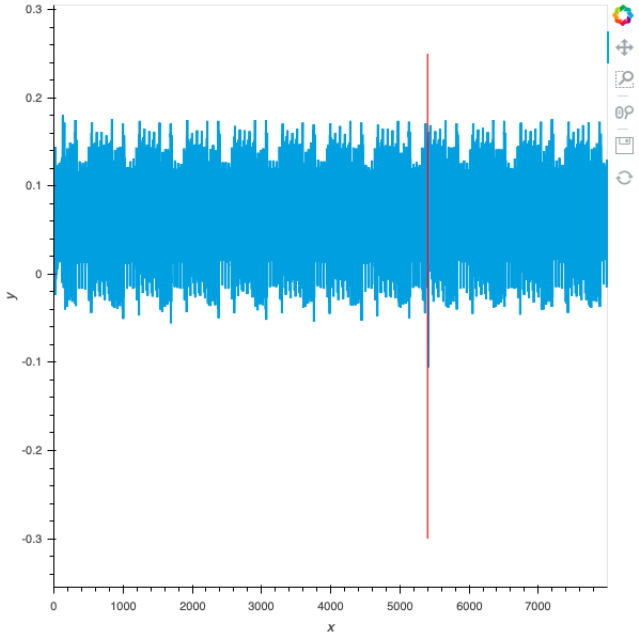

Figure 67: Glitching the AES implementation.

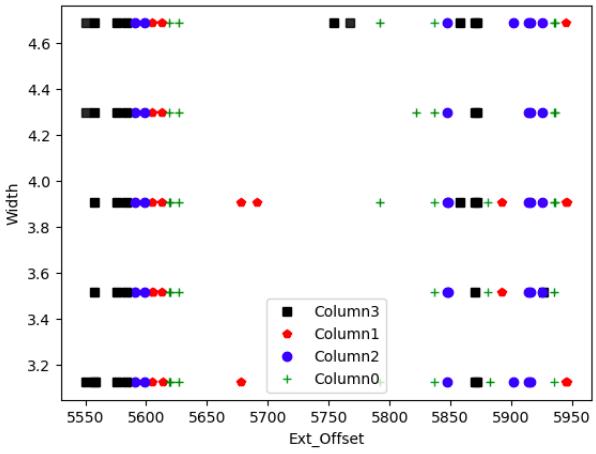

Figure 68: Campaign Results: Successful Column Glitches.

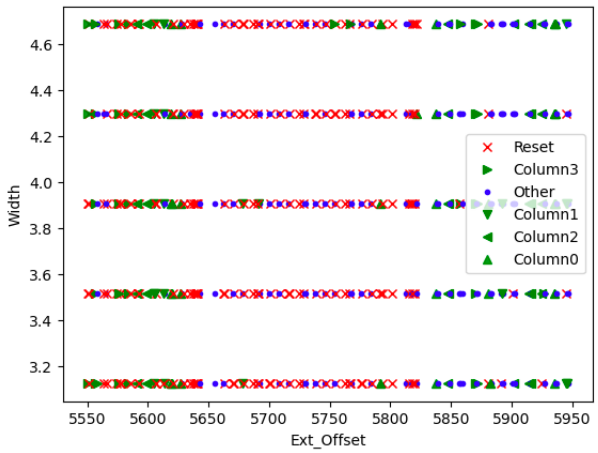

Figure 69: Campaign Results: Reset & Other Obtained From The Campaign.

3.10.3 Conclusion

We can observe from the results that through successful glitching of 8^{th} column of the AES, we were able to reveal the actual key that we are using to encrypt our plaintext. Unlike DPA & CPA we only use a single trace to break the AES algorithm. This makes the DFA a more formidable foe.

4 COUNTERMEASURES TO SIDE CHANNEL ATTACKS

Side-channel leakage has been a known threat to secure electronic designs for years, yet the evaluation and mitigation of side-channel leakage has remained an ad hoc process during most of that time. Design tools, which are essential to master the design complexity of modern electronic systems, are still rudimentary in handling side-channel leakage mitigation. There are several reasons for that. First, side-channel leakage is not a singular security issue, but a collection of implementation effects in a secure system. Each of these may be handled differently, using different tools, depending on the nature of the leakage and the abstraction level of design. Secondly, side channel analysis and mitigation are evolving concepts. Solutions that work today may not work tomorrow, so the effort of developing a tool today may not pay off. Finally, countermeasures suffer from poor compatibility. For example, countermeasures against fault attacks and side-channel attacks have been shown to be incompatible.

In this section, we will be exploring two different techniques to prevent side channel attacks from revealing the key in the AES implementation. To implement, we will be modifying the unprotected AES implementation.

4.1 Random Delay Model

Insertion of random delays in the execution flow of a cryptographic algorithm is a simple yet rather effective countermeasure against side-channel and fault attacks. To our knowledge, random delays are widely used for the protection of cryptographic implementations in embedded devices, mainly smart cards. It belongs to a group of hid countermeasures that introduce additional noise (either in time, amplitude,

or frequency domain) to the side channel leakage while not eliminating the informative signal itself. This is in contrary to masking countermeasures that eliminate the correlation between the side channel leakage and the sensitive data processed by an implementation.

Hiding countermeasures increases the complexity of attacks while not rendering them completely impossible. They are not treated in academia as extensively as masking but are of great importance in industry. A mixture of multiple hiding and masking countermeasures would often be used in a real-life protected implementation to raise the complexity of attacks above the foreseen capabilities of an adversary.

In my experiment, we use the clock function in the C library "time.h" to generate random delays after each stage of the AES for a particular subkey byte. We use the following algorithm to add delay: After the delay calculation, we assign a random

Algorithm 7: Delay Function.

Result: Delay Calculated
Input: time in milliseconds, current timestamp, int i;
while *present time¡current timestamp+time in miliseconds* **do**
  increment i;
end

delay value to the rand() function with 1000 and later timestamps taken before each round of AES, as the seed value for the rand function. We perform the same attack again and observe the results.

4.1.1 Results - Differential Power Attacks rand()%1000

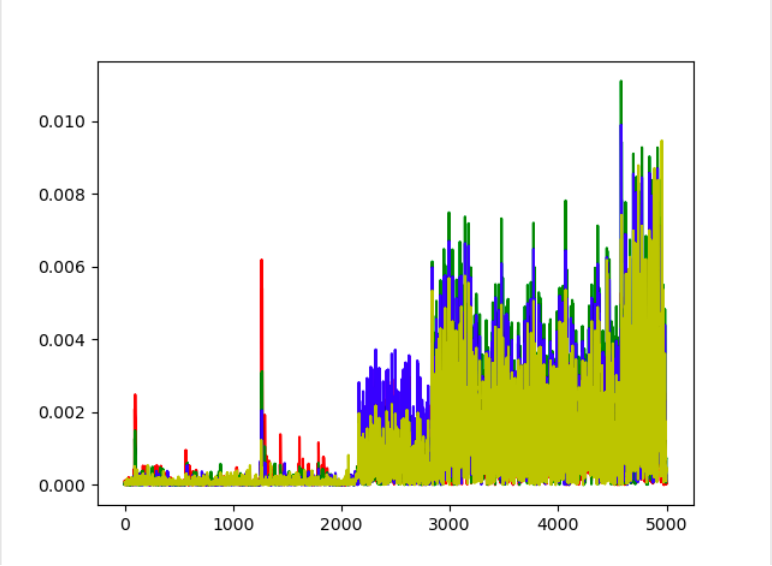

Figure 70: Differential Power Analysis On 0^{th} SubKey: Actual Subkey(Red) vs 5 Most Likely Key Guesses During 1^{st} Run.

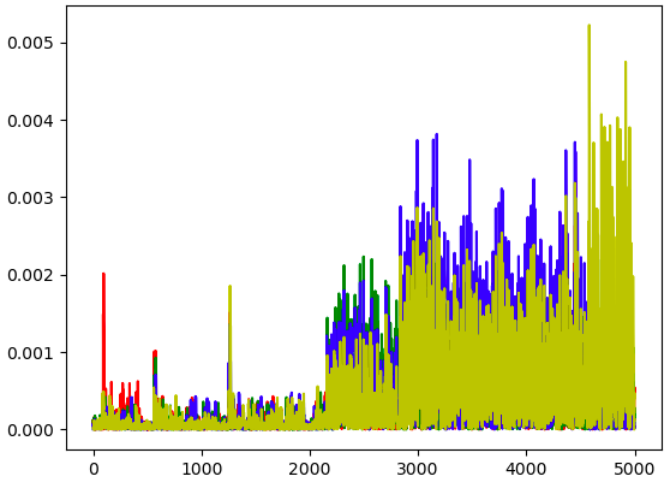

Figure 71: Differential Power Analysis On 1^{st} SubKey: Actual Subkey(Red) vs 5 Most Likely Key Guesses During 1^{st} Run.

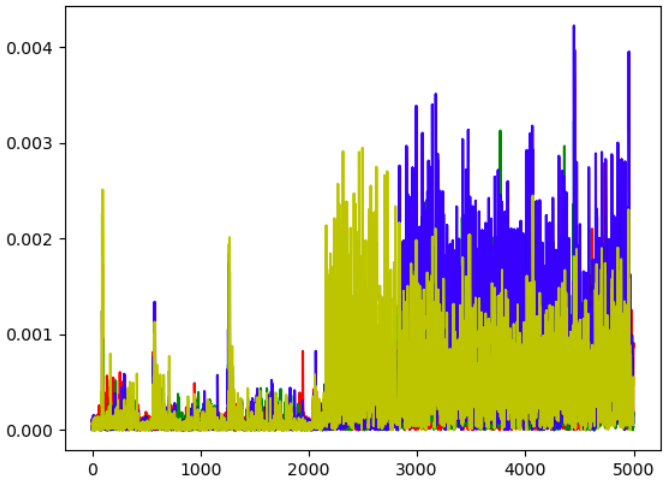

Figure 72: Differential Power Analysis On 2^{nd} SubKey: Actual Subkey(Red) vs 5 Most Likely Key Guesses During 1^{st} Run.

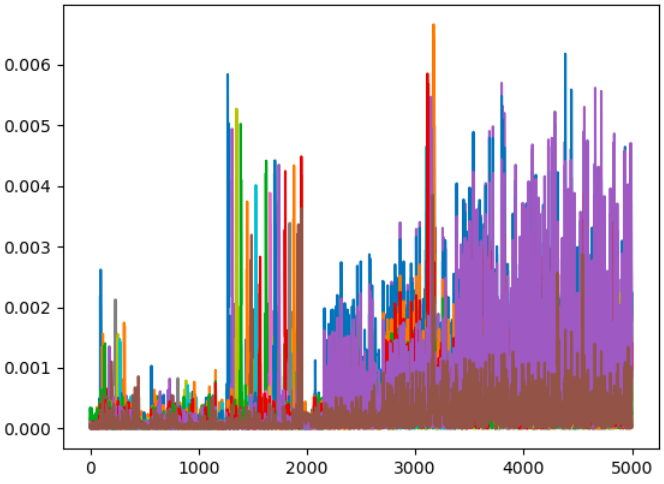

Figure 73: Differential Power Attack On All Bytes After Adding Random Delays.

4.1.2 Results - Differential Power Attacks, Delay - rand()%(timestamp%10000)

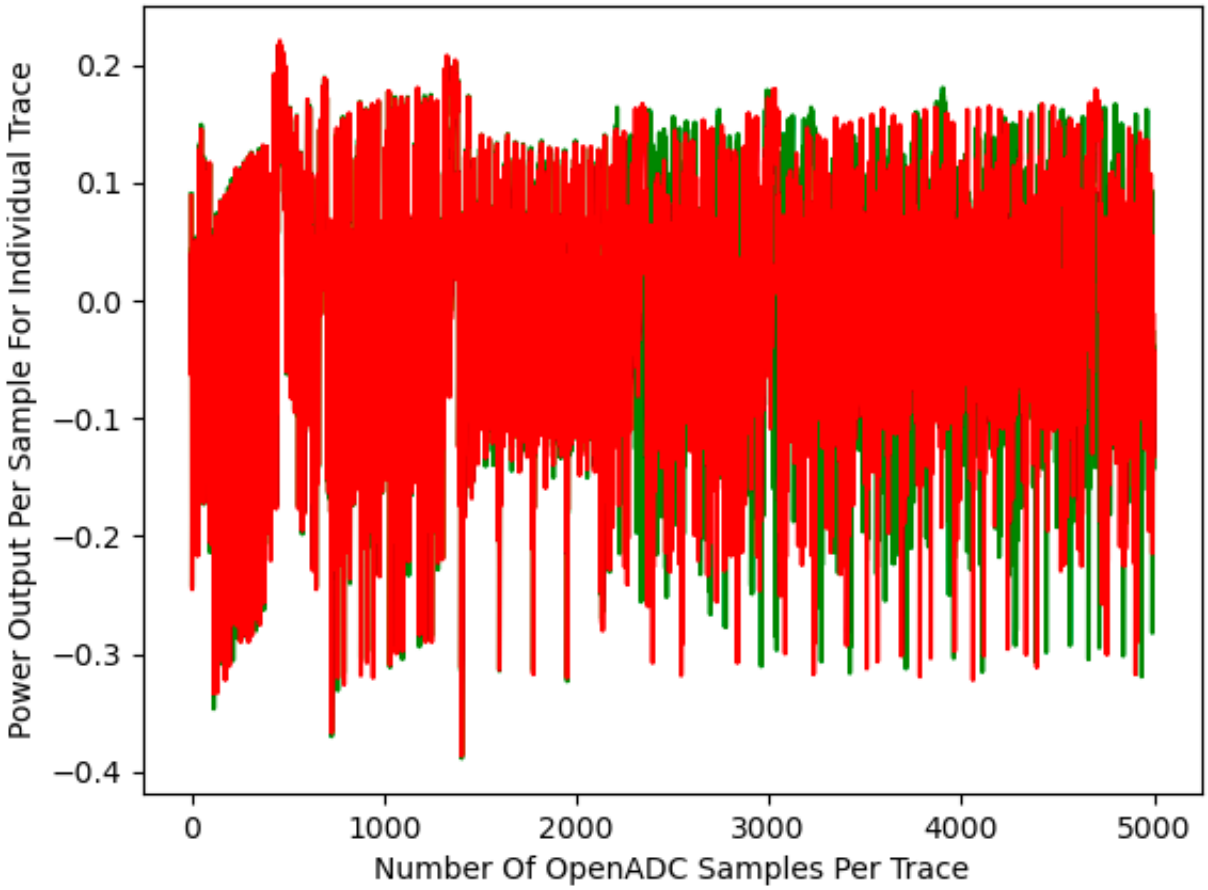

Figure 74: Differential Power Analysis On 1^{th} Byte For Adding Random Delays During 1^{st} Run.

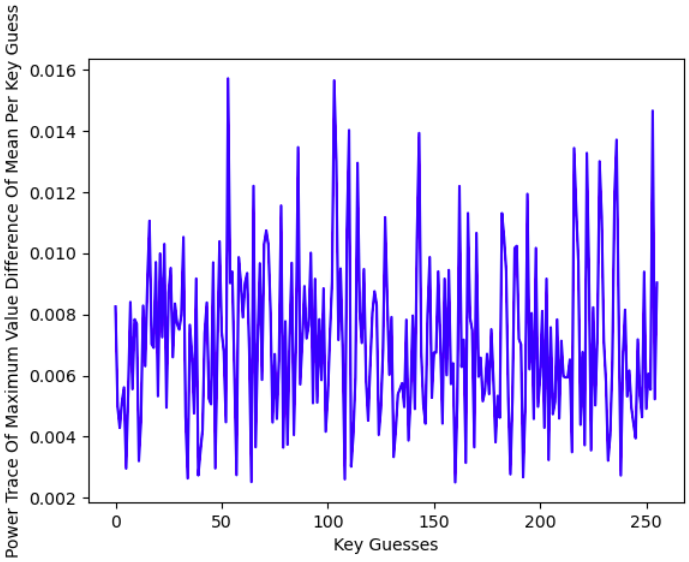

Figure 75: Differential Power Attack On 1^{th} byte After Adding Random Delays During 1^{th} Run.

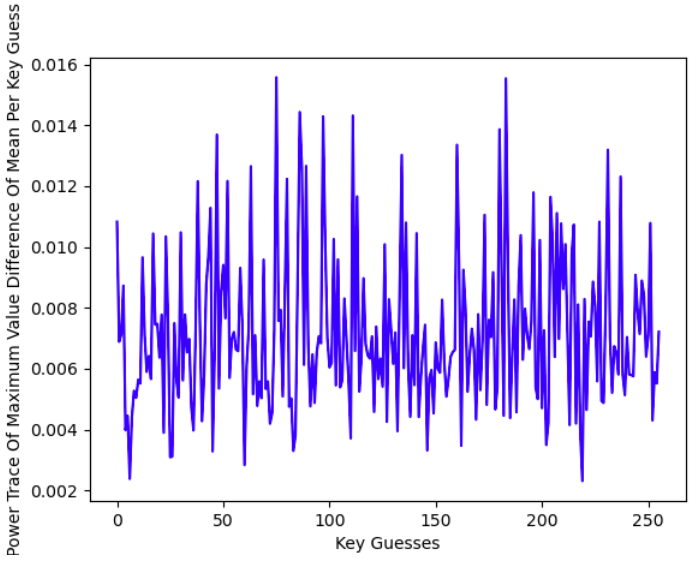

Figure 76: Differential Power Attack On 2^{nd} byte After Adding Random Delays During 2^{nd} Run.

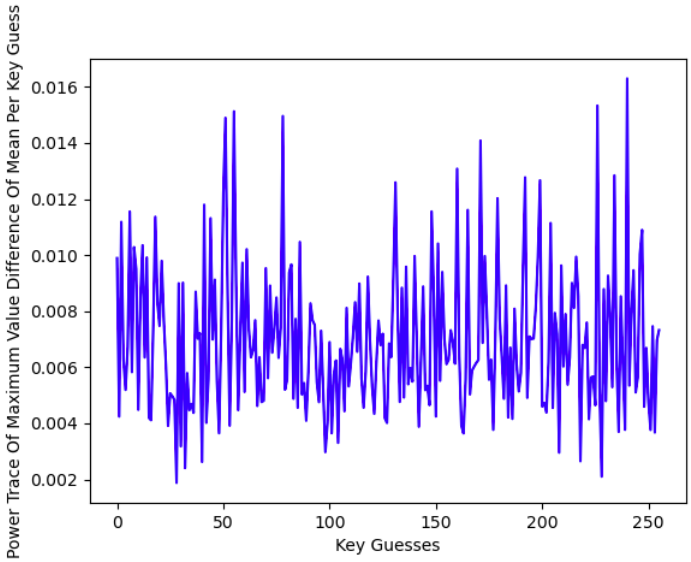

Figure 77: Differential Power Attack On 3^{rd} byte After Adding Random Delays During 3^{rd} Run.

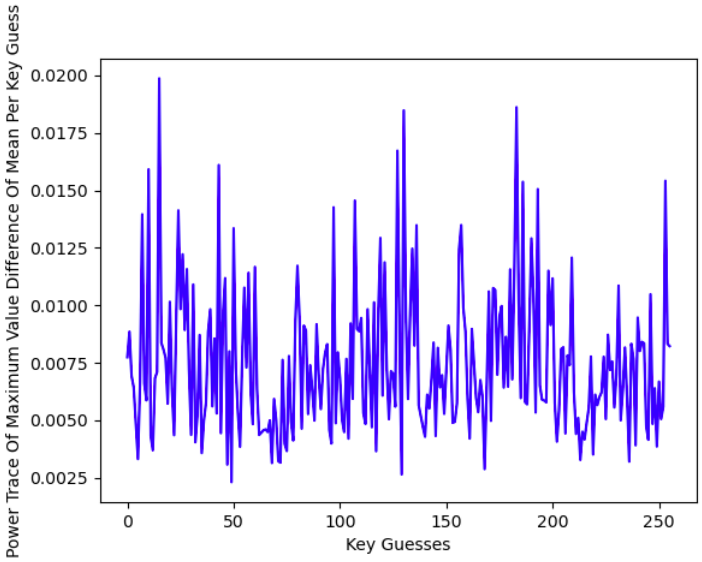

Figure 78: Differential Power Attack On The 4^{th} byte After Adding Random Delays During 4^{th} Run.

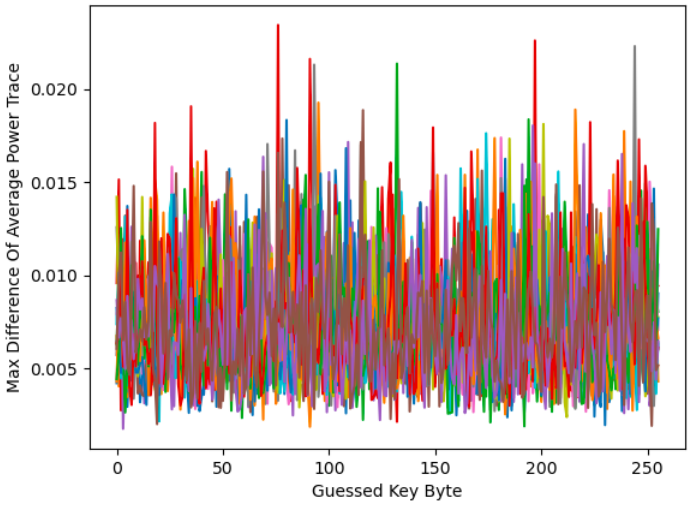

Figure 79: Differential Power Attack On All Bytes After Adding Random Delays During 1^{st} Run.

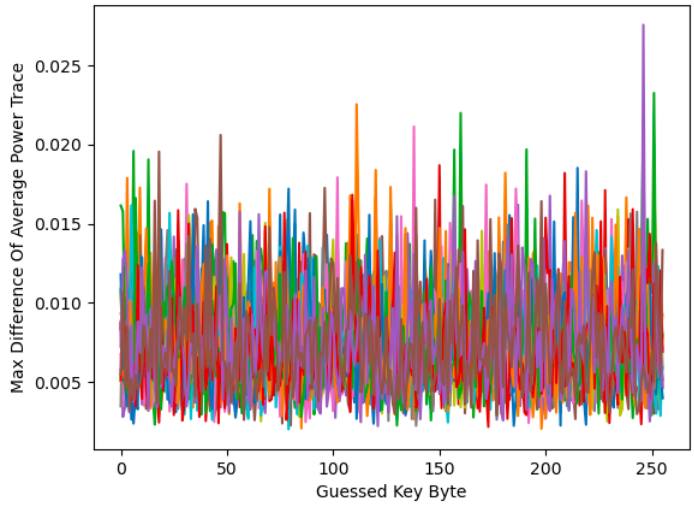

Figure 80: Differential Power Attack On All Bytes After Adding Random Delays During 2^{nd} Run.

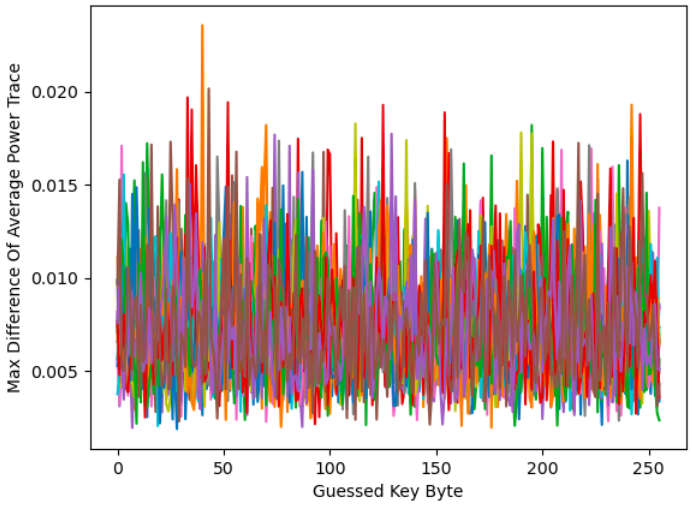

Figure 81: Differential Power Attack On All Bytes After Adding Random Delays During 3^{rd} Run.

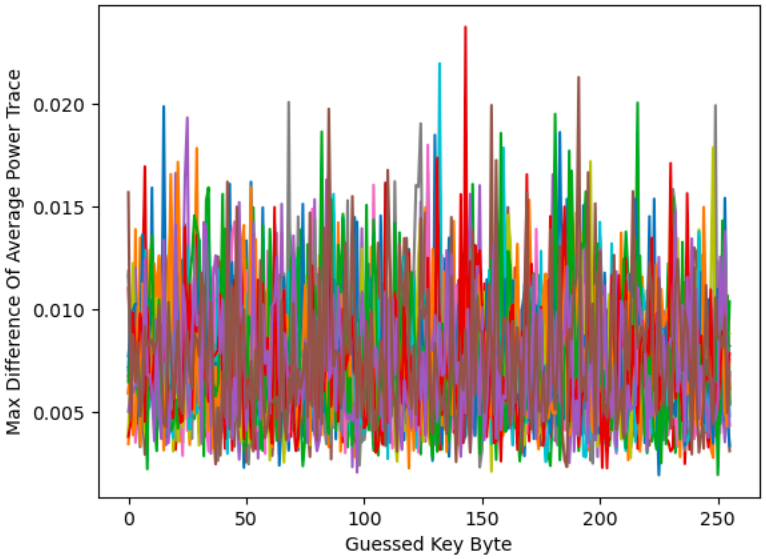

Figure 82: Differential Power Attack On All Bytes After Adding Random Delays During 4^{th} Run.

4.1.3 Results - Correlation Power Attacks, Delay - rand()%(timestamp%10000)

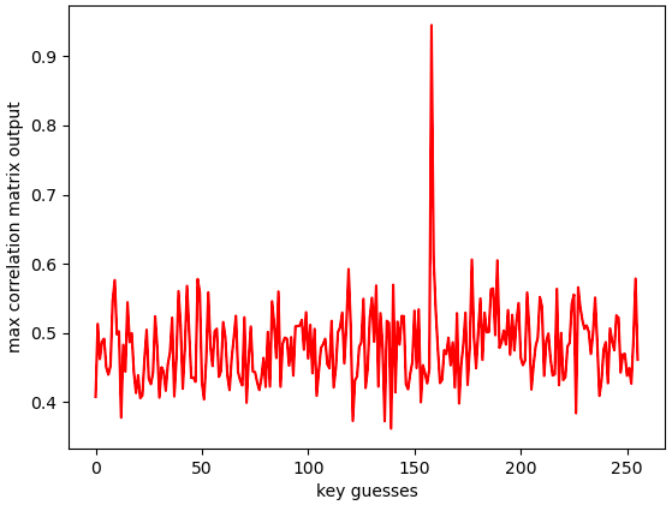

Figure 83: Correlation Power Attack Results After Addition Of Random Delays.

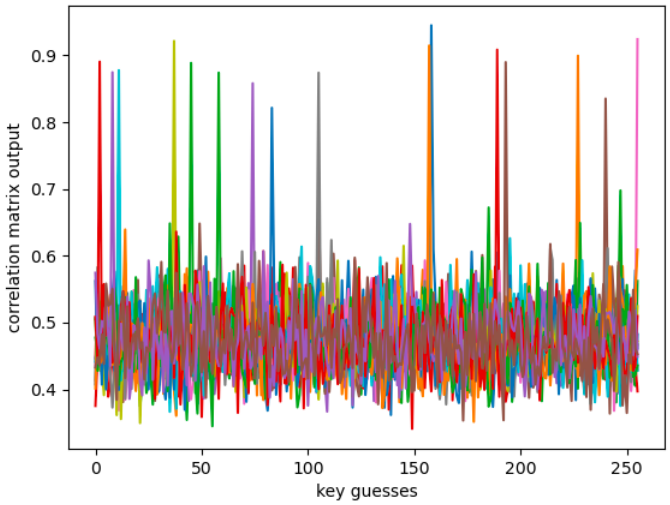

Figure 84: Correlation Power Attack Results After Addition Of Random Delays.

4.1.4 Results - Differential Fault Attacks, Delay - rand()%(timestamp%10000)

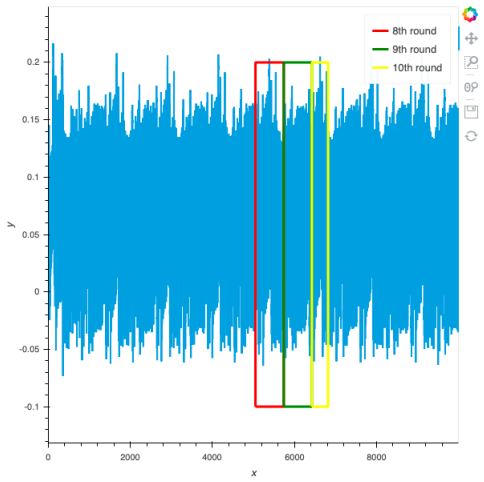

Figure 85: Power Trace Of AES Execution With Random Delays.

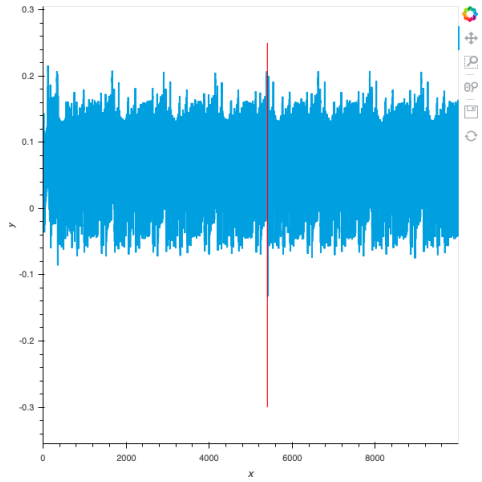

Figure 86: Glitching AES Execution With Random Delays.

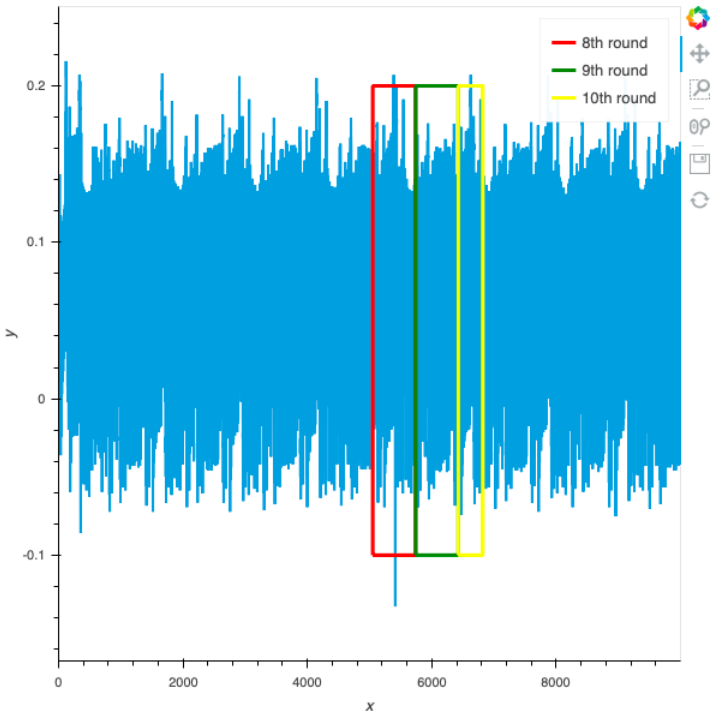

Figure 87: AES Faulty Outputs For 8^{th}, 9^{th} & 10^{th} Rounds.

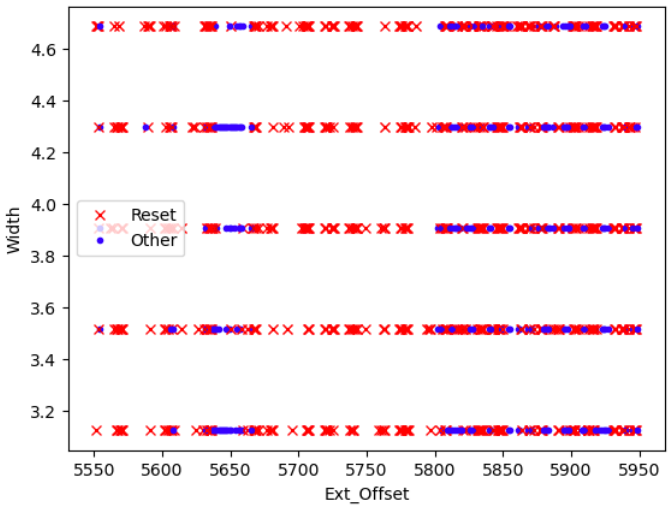

Figure 88: Reset & Other Resulting Plot From The Campaign.

4.1.5 Conclusions

When we add random delays for each stage of AES execution, we can see that it increases the noise level in the voltage signal being captured during power trace capture. This leads to incorrect peak formation during differential power attacks as seen above. The number of key guesses that have a high DOM value increases, which leads to incorrect deductions and thus incorrect key guess. We tried to use a fixed seed value for the random function in the first case and then used the random seed value based on the timestamp during which the algorithm is executing and we can see an increase in noise.

For differential fault analysis, the random delays lead to unsuccessful clock glitching for the last three columns of AES. The reason is that the glitching happens at the incorrect instance when the power trace is being captured by the delay rather than the mix column key stage of AES algorithm. Thus, when compared with correct, the golden cipher does not lead to any correlation between both. The only the glitches we get causes a reset, which does not aid us in deciphering the actual key.

4.1.6 Limitations

This attack works well to prevent differential power attack and differential fault attack. However, it does not prove effective against correlation power attacks as seen from the results. The delay does not effect the covariance between the Hamming weight and the power trace gathered during the attack in any way. There is, however, a possibility of adding delays using a true random number generator and using random algorithm execution to increase the noise to such a level that the correlation power attacks become unsuccessful.

4.2 Dummy Sbox Model

We learnt in Section 3.5 how Sbox is one of the crucial aspects for a successful attacker. For the attack to be successful, the more the number of bits that are same between the hypothetical and the actual leakage, the better are the results for the maximum difference of mean and maximum correlation matrix output. This is possible if the internal values of the AES computation match with our hypothetical AES internal values for different key guesses. Thus, the Sbox should be exactly same.

In this countermeasure mode, each time we perform the attack, the AES cipher is computed using a random AES Sbox. This changes the internal values that we are trying to match with during our attack and hence our attack should be unsuccessful.

We use the same "time.h" C library to compute a value between 0 & 255 and randomly place them in the Sbox matrix each time we gather power traces for a particular number of iterations for the attack. As soon as the attacker tries to perform the attack again, the Sbox resets to a different value altogether. Thus preventing any leakage of the key. Therefore, even if the attacker is successful in leaking the sbox implementation.

We started my experiments by performing column operations on the existing Sbox and the results that we got were mixed bad, of good, and bad. The dummy sbox is used to work for differential power analysis still, but for correlation power analysis is only used to work for a small number of power trace captures. For higher values, the key guess was relatively close to the actual key. Thus, we gave up using a fixed sbox and thought of using a variable dummy Sbox instead.

4.2.1 Results - Differential Power Attacks

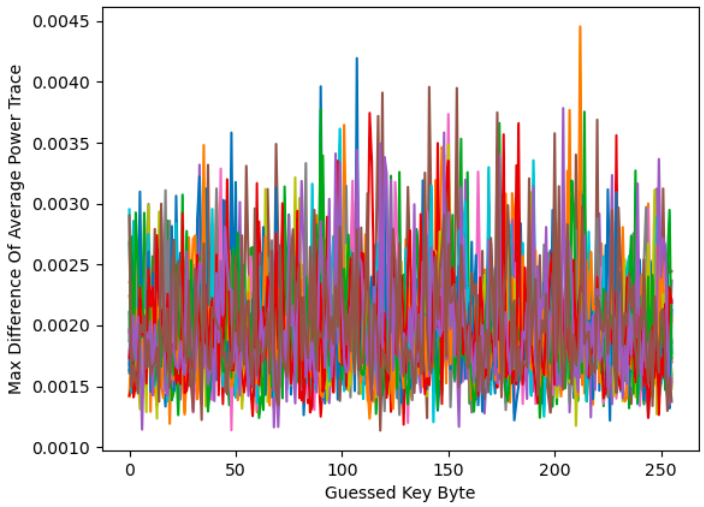

Figure 89: Difference Of Mean Plot For DPA On Dummy Sbox AES Implementation 1.

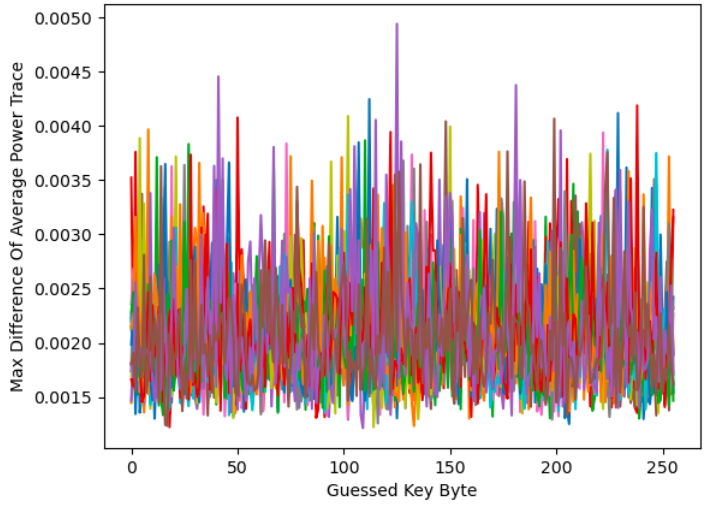

Figure 90: Difference Of Mean Plot For DPA On Dummy Sbox AES Implementation 2.

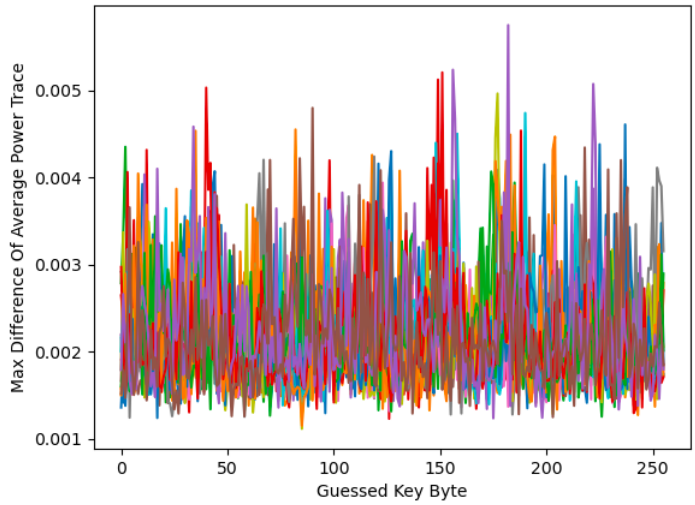

Figure 91: Difference Of Mean Plot For DPA On Dummy Sbox AES Implementation 3.

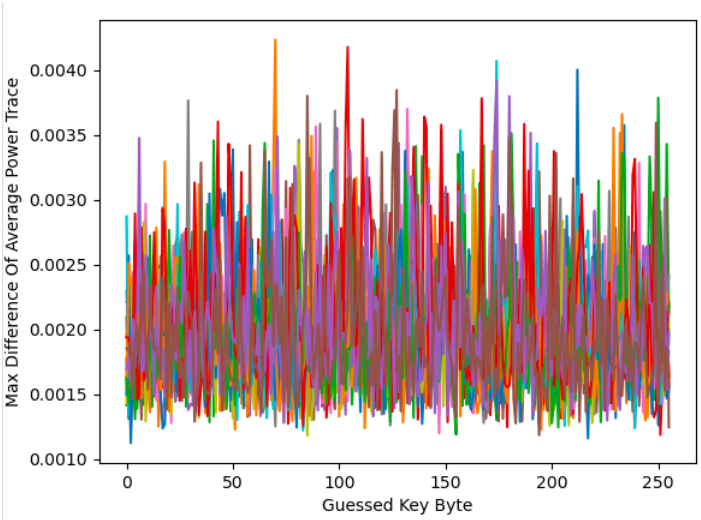

Figure 92: Difference Of Mean Plot For DPA On Dummy Sbox AES Implementation 4.

4.2.2 Results - Correlation Power Attacks

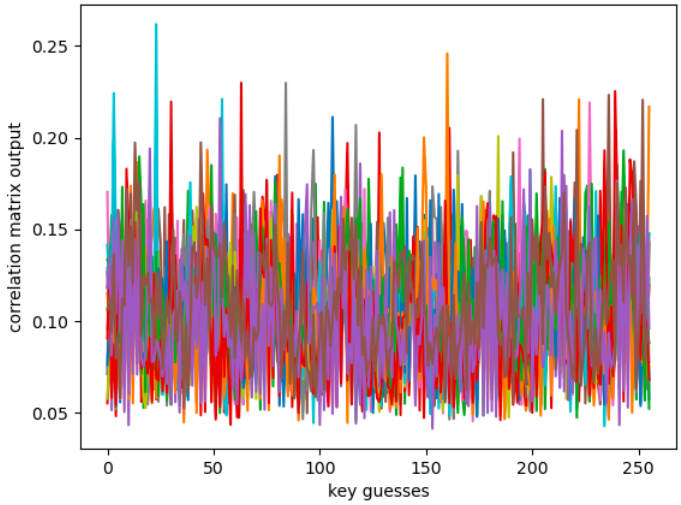

Figure 93: Difference Of Mean Plot For CPA On Dummy Sbox AES Implementation 1 For 5000 Power Traces.

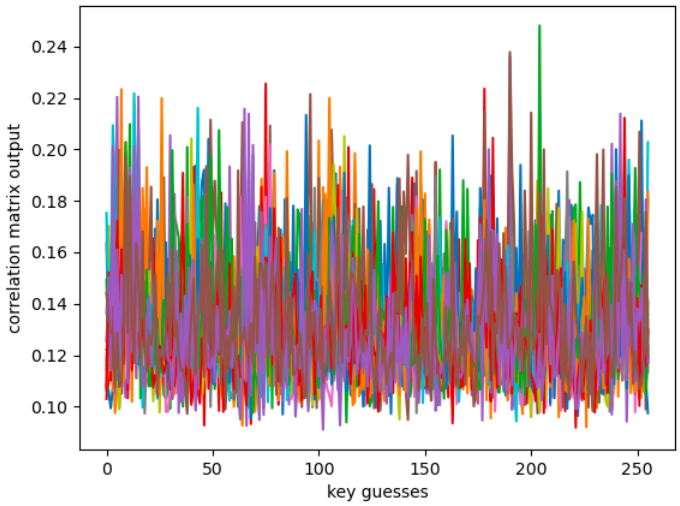

Figure 94: Difference Of Mean Plot For CPA On Dummy Sbox AES Implementation 2 For 1000 Power Traces.

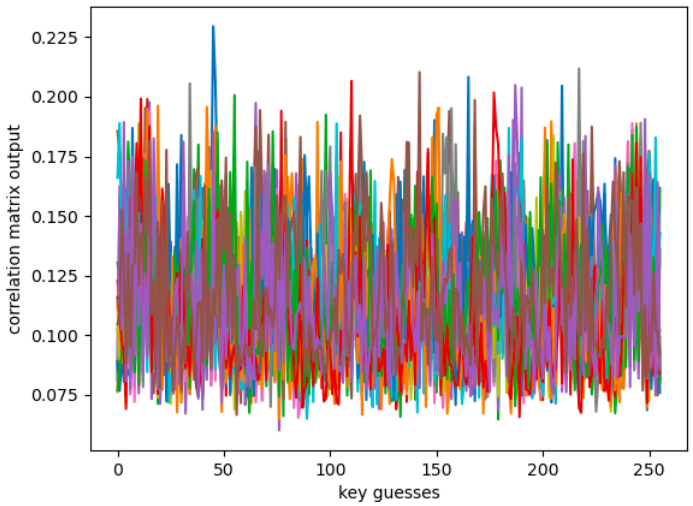

Figure 95: Difference Of Mean Plot For CPA On Dummy Sbox AES Implementation 3 For 2000 Power Traces.

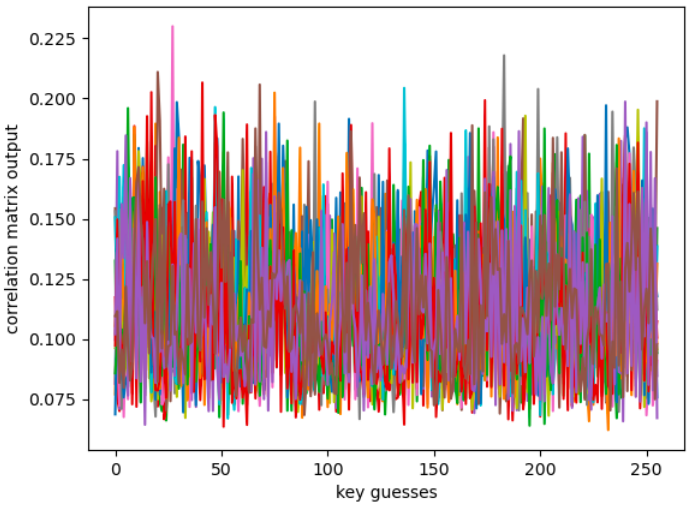

Figure 96: Difference Of Mean Plot For CPA On Dummy Sbox AES Implementation 4 For 2000 Power Traces.

4.2.3 Results - Differential Fault Analysis

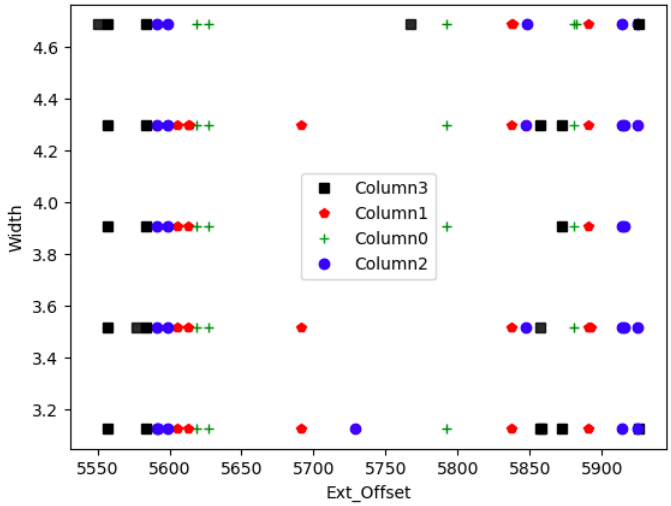

Figure 97: Campaign Results - Successful Column Glitches Plot.

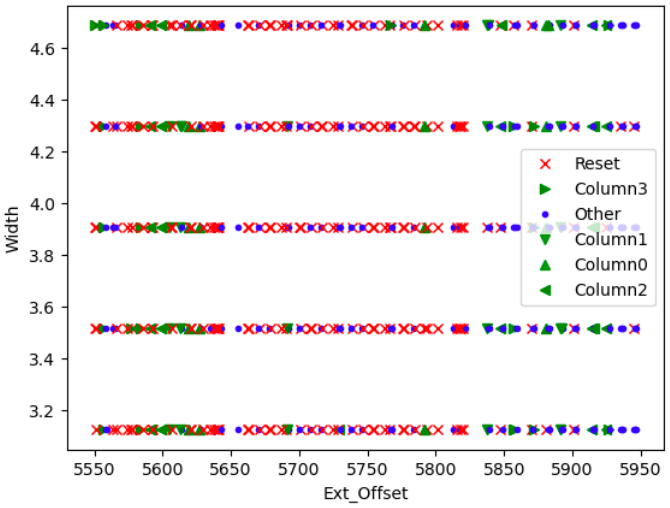

Figure 98: Campaign Results - All Glitch Attempts.

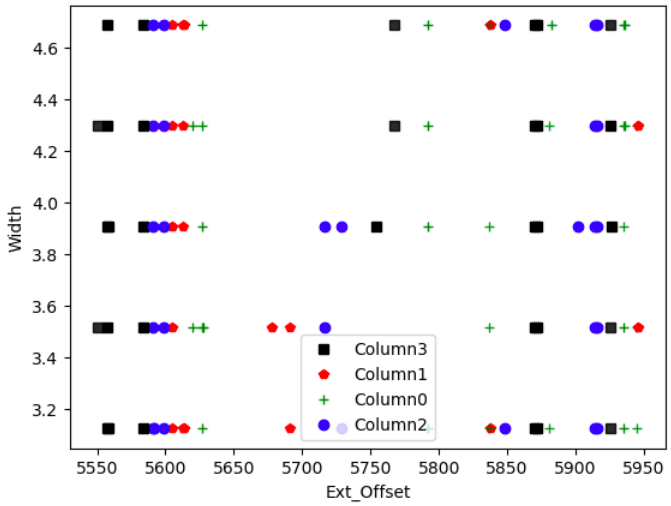

Figure 99: Campaign Results: Successful Column Glitches Plot.

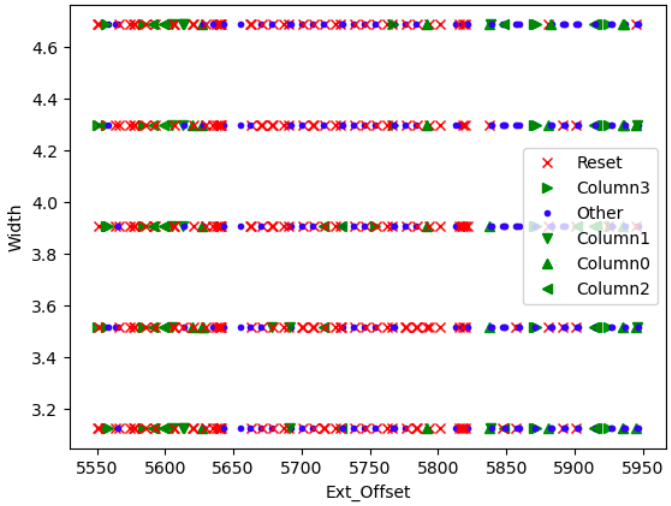

Figure 100: Campaign Results - All Glitch Attempts.

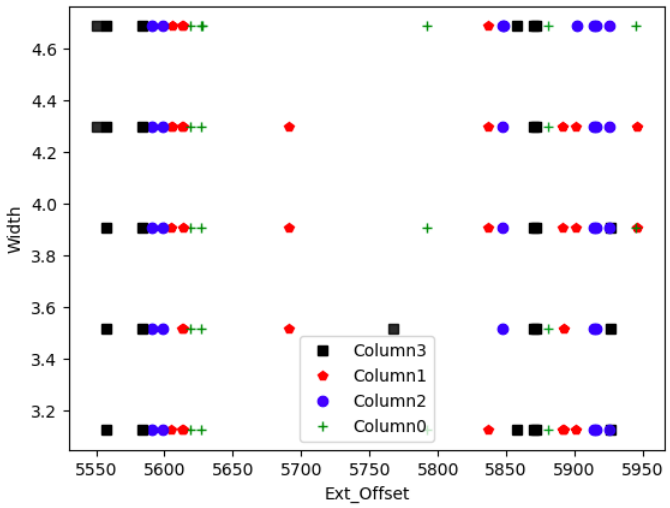

Figure 101: Campaign Results: Successful Column Glitches Plot.

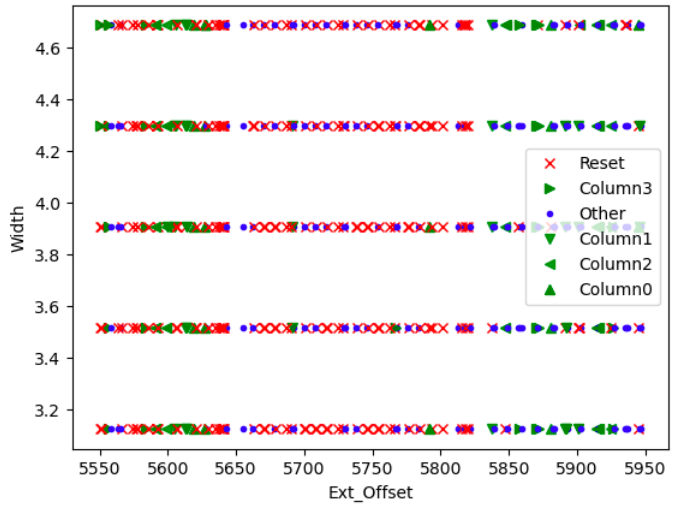

Figure 102: Campaign Results: All Glitch Attempts.

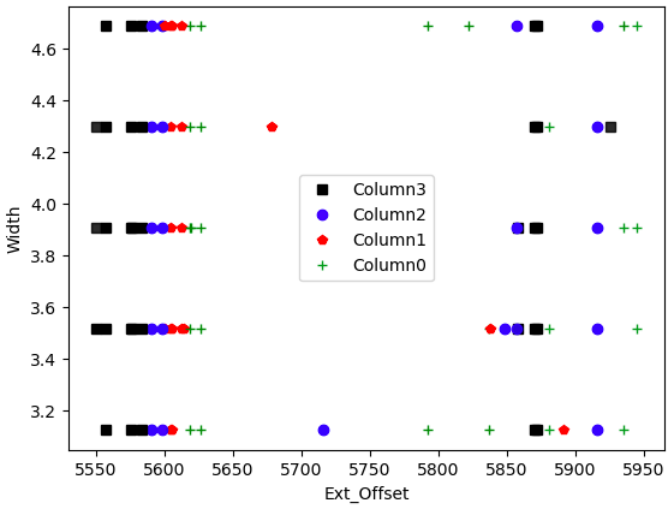

Figure 103: Campaign Results: Successful Column Glitches Plot.

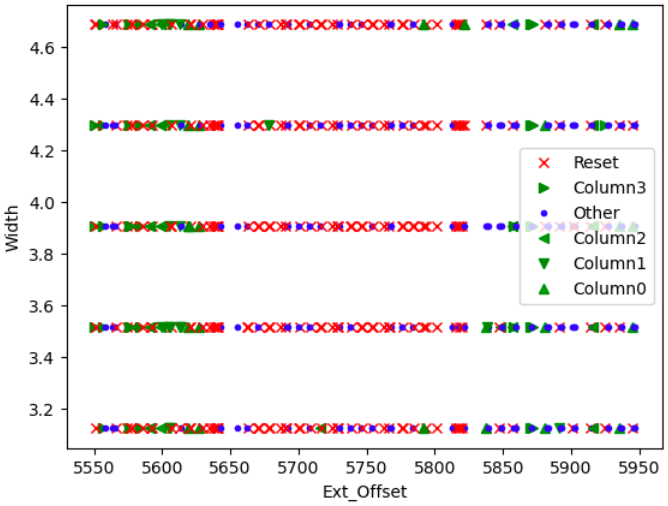

Figure 104: Campaign Results: All Glitch Attempts.

4.2.4 Conclusion

The countermeasure is successful in helping us prevent the attack. We can see from the results that even if we increase the number of power traces collected, the plot that we get for key guesses vs the maximum difference of the mean or maximum correlation value is completely random with no signs of any patterns being formed that might give the attacker any clue to the actual value of the key.

For differential fault analysis, even though we do see successful glitches for columns 8, 9 & 10 we are not able to obtain any possible value of the key either from column 9 or column 10.

4.2.5 Limitations

While the dummy sbox model works flawlessly for all three kinds of attacks, it does come with its fair share of limitations. The sbox that is used in the original cipher generates a cipher which can only be deciphered later if we use the correct inverse Sbox matrix. That can be a tedious task since the correct multiplicative inverse Sbox may not exist for each dummy sbox as the inverse of a matrix may or may not exist. The inverse sbox is always calculated using the formula $GF(2^8) = \frac{GF(2)[x]}{(x^8+x^4+x^3+x+1)}$. This can cause incorrect plain text calculations and thus causing a problem for the user.

5 CONCLUSION

In this section we will summarize the content of our thesis and discuss the research problems that we solved through our research. Later, we discuss further directions for future work.

In our thesis, we observed the current problems that plague the unprotected AES implementations and what measures can be taken to make them more resilient to side channel attacks. Advanced Encryption Standard is the current standard for block ciphers and is used in smart card embedded security applications. Side channels are always present as a byproduct of the performance analysis of any hardware implementation of an algorithm. However, in the case of cryptographical algorithms, they can leak vital & sensitive information which the user wants to protect.

In Chapter 3, we see in how many ways we can leak data using sophisticated methods involving data analysis and fault injection. We examined the relationship between power and the data being serially written in the device and exploited this aspect to break AES in Differential Power Attacks. Later, we performed a more advanced version of the attack: Correlation Power Attack. We used Hamming weight model for implementing CPA as we saw that there is a direct proportionality between the power and the Hamming weight of the data being written. We observed how more power efficient and effective attack was compared to DPA.

Moving further, we explored the possibility of adding faults or glitches to different aspects of the microcontroller. We explored voltage and clock glitching and compared the effectiveness in giving us consistent results after performing repeatedly. We used this technique to perform differential fault attacks, where we inserted a fault into a particular round of the AES and compared that to a round that was normally executed to reveal the actual key. This is a different class of attack in the sense that as an attacker we do not need to compare the hypothetical leakage with the actual

leakage and thus can be implemented with any prior knowledge of the sbox of AES. This makes it a much more formidable attack.

However, it is quite challenging to reveal the actual key as we need to know the exact glitch width and offset to perform the attack. We need to perform lots of experiments to determine those values. Even if the attacker doesn't get the actual key, he can still tamper with the device causing malfunction while performing this attack.

The optimization and modification of an unprotected AES specially adapted for the needs of the industry is necessary. In Section 4, we explore some set of possible ways of preventing these attacks. We implemented two models: *random delay* and *dummy sbox*.

In random delay model, we observed that by adding random delays it is a great way to introduce more noise in the power trace. This tampers with the difference of mean between the hypothetical and actual leakage since the hypothetical leakage is based on the regular implementation of AES. This countermeasure also proved a hindrance for differential fault attacks since the glitch placement and width were decided based on the functioning of unprotected AES. Since we are using the clock glitch, by modifying the time of execution, we are unable to perform the attack successfully unless we calibrate the glitch parameters according to the modified attack. Moreover, the delay added each time we perform the attack is going to be random, so we will not be successful in implementing the attack. This model is unsuccessful in case of correlation power attack as it does nothing to hinder the linear relationship between Hamming weight and power. This made us search for alternative countermeasures for the attacks.

In the next subsection, we proposed a dummy sbox model for preventing AES. It proved successful in preventing key leakage for all three types of attacks. We already

knew that the internal value of AES implementation is exploited by the attacker, so coming up with modifying the internal values was a natural solution. The attack was extremely successful in preventing CPA even with a large number of power traces gathered during the attack.

To summarise, we were able to achieve what we proposed to achieve through our research. We were able to successfully modify the cryptographic implementation to make it resistant to side channel attacks on a reconfigurable hardware. The implementation can be used as a plug and play solution which can replace vulnerable unprotected AES implementations. We predict and hope that in the near future more powerful countermeasures will be proposed to make our every day devices more secure for personal use.

References

[1] Boey, K. H., Lu, Y., O'Neill, M., and Woods, R. Random clock against differential power analysis. In *2010 IEEE Asia Pacific Conference on Circuits and Systems* (2010), pp. 756–759.

[2] Bow, I., Bete, N., Saqib, F., Che, W., Patel, C., Robucci, R., Chan, C., and Plusquellic, J. Side-channel power resistance for encryption algorithms using implementation diversity. *Cryptography 4*, 2 (2020).

[3] Bulck, J. V., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T. F., Yarom, Y., and Strackx, R. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)* (Baltimore, MD, Aug. 2018), USENIX Association, p. 991–1008.

[4] Canella, C., Genkin, D., Giner, L., Gruss, D., Lipp, M., Minkin, M., Moghimi, D., Piessens, F., Schwarz, M., Sunar, B., Bulck, J., and Yarom, Y. Fallout: Leaking data on meltdown-resistant cpus. In *CCS 2019 - Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (11 2019), ACM/IEEE, pp. 769–784.

[5] Coron, J.-S., and Kizhvatov, I. An efficient method for random delay generation in embedded software. In *Cryptographic Hardware and Embedded Systems - CHES 2009* (Berlin, Heidelberg, 2009), C. Clavier and K. Gaj, Eds., Springer Berlin Heidelberg, pp. 156–170.

[6] Hettwer, B., Das, K., Leger, S., Gehrer, S., and Güneysu, T. Lightweight side-channel protection using dynamic clock randomization. In *30th International Conference on Field-Programmable Logic and Applications, FPL 2020, Gothenburg, Sweden, August 31 - September 4, 2020* (2020), N. Mentens, L. Sousa, P. Trancoso, M. Pericàs, and I. Sourdis, Eds., IEEE, pp. 200–207.

[7] Kocher, P., Jaffe, J., and Jun, B. Differential power analysis. In *Advances in Cryptology — CRYPTO' 99* (Berlin, Heidelberg, 1999), M. Wiener, Ed., Springer Berlin Heidelberg, pp. 388–397.

[8] Mangard, S., Oswald, E., and Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag, Berlin, Heidelberg, 2007.

[9] Masoomi, M., Masoumi, M., and Ahmadian, M. A practical differential power analysis attack against an fpga implementation of aes cryptosystem. In *2010 International Conference on Information Society* (2010), pp. 308–312.

[10] MIKAMI, S., YOSHIDA, H., WATANABE, D., and SAKIYAMA, K. Correlation power analysis and countermeasure on the stream cipher enocoro-128v2. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E96.A*, 3 (2013), 697–704.

[11] Mukhopadhyay, D. In *Hardware Security: Design, Threats, and Safeguards*. CRC Press, 2015, ch. 1, pp. 30–31.

[12] O'Flynn, C. ChipWhisperer 5.5, 2021. [Online; accessed 11-April-2021].

[13] O'Flynn, C. CW1101 ChipWhisperer-Nano, 2021. [Online; accessed 10-April-2021].

[14] Philip, M. A., and Vaithiyanathan. A survey on lightweight ciphers for iot devices. In *2017 International Conference on Technological Advancements in Power and Energy (TAP Energy)* (2017), pp. 1–4.

[15] Shannon, C. E. Communication theory of secrecy systems. *The Bell System Technical Journal 28*, 4 (1949), 656–715.

[16] Skorobogatov, S. Hardware security evaluation of max 10 fpga, 2019.