Computer Engineering Lab
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2019

Quantum & Computer Engineering

MSc THESIS

S-Net, A Neural Network Based Countermeasure for AES

Pradeep Venkatachalam

Q&CE-CE-MS-2019-20

Abstract

Hardware implementations of encryption schemes are unprotected against side-channel analysis techniques. Physical realizations of secure algorithms leak side-channel information through power, noise, time, sound and electromagnetic radiation. Data-dependent correlations with this leakage are exploited to obtain secret information. Power analysis techniques are powerful, undetectable and non-intrusive attacks that allow an adversary to extracts the secret key of the encryption scheme. These techniques rely on analyzing the power consumed by these physical realizations using leakage models and statistical techniques.

Implementing a countermeasure against power analysis attacks require a thorough understanding of the attack, encryption algorithm and it's implementation on hardware and software. Conventional countermeasures for AES against power analysis techniques minimize the side-channel information by implementing masking and hiding strategies at different abstraction levels. This thesis investigates a new class of countermeasures known as "breaking" through the implementation of the Substitution Box transformation using a neural network (S-Net). The inherent properties associated with the neural network architecture is expected to remove the correlation between the power consumed and the secret key used for encryption by breaking the linear power characteristics assumed by the leakage model.

The proposed approach was implemented in software and an attack framework is used to run side-channel attacks and quantify information leakage. The effectiveness of the implemented countermeasure is measured by checking and quantifying it's security against Differential and Correlation Power Analysis, Template and Deep Learning based techniques. The results indicate that the implementation is secure against these attacks.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

S-Net, A Neural Network Based Countermeasure for AES

by

Pradeep Venkatachalam
Computer Engineering
Student Number: 4735323
Email: pvenkatachalam@student.tudelft.nl

Course : MSc Thesis
Supervising Professor : Dr. Ir. M. Taouil, CE, TU Delft
Chairperson : Prof. dr. ir. S. Hamdioui, CE, TU Delft
First Member : Dr. Ir. R. van Leuken, CAS, TU Delft
Second Member : Ir. Abdullah Aljuffri, CE, TU Delft
Duration : December 1, 2018 - August 30, 2019

This work was performed in:

Computer Engineering Lab
Quantum and Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

S-Net, A Neural Network Based Countermeasure for AES

by Pradeep Venkatachalam

Abstract

Hardware implementations of encryption schemes are unprotected against side-channel analysis techniques. Physical realizations of secure algorithms leak side-channel information through power, noise, time, sound and electromagnetic radiation. Data-dependent correlations with this leakage are exploited to obtain secret information. Power analysis techniques are powerful, undetectable and non-intrusive attacks that allow an adversary to extracts the secret key of the encryption scheme. These techniques rely on analyzing the power consumed by these physical realizations using leakage models and statistical techniques.

Implementing a countermeasure against power analysis attacks require a thorough understanding of the attack, encryption algorithm and it's implementation on hardware and software. Conventional countermeasures for AES against power analysis techniques minimize the side-channel information by implementing masking and hiding strategies at different abstraction levels. This thesis investigates a new class of countermeasures known as "breaking" through the implementation of the Substitution Box transformation using a neural network (S-Net). The inherent properties associated with the neural network architecture is expected to remove the correlation between the power consumed and the secret key used for encryption by breaking the linear power characteristics assumed by the leakage model.

The proposed approach was implemented in software and an attack framework is used to run side-channel attacks and quantify information leakage. The effectiveness of the implemented countermeasure is measured by checking and quantifying it's security against Differential and Correlation Power Analysis, Template and Deep Learning based techniques. The results indicate that the implementation is secure against these attacks.

Contents

List of Figures

Acknowledgements

I would like to take this opportunity to express the deepest gratitude to thank all the people who have supported me through this thesis. Without the guidance and persistent support of these people, this dissertation would have been far from completion.

I would firstly like to thank my supervisor Dr. Ir. M Taouil who demonstrated and inculcated in me an exciting spirit of adventure in exploring this fascinating field of hardware cryptography. I would also like to thank Prof. dr. ir. S.Hamdioui for introducing me into the department through the wonderful lectures. I would also like to thank Ir. Abdullah Aljuffri for his constant support and guidance through the project.

Every thesis requires strong fundamentals, and for this, I thank Dr. M.R. Arulalan Rajan and Dr. Zeki Erkin for building the foundation required to enter this field. A special mention to Abhairaj, Siddharth and Surya for keeping me company on long days.

Last and definitely not the least, I would like to thank Swaroopini Ramachandran for the constant source of love, support, and inspiration through these two years.

1 Introduction

This chapter introduces the subject addressed in the thesis, the motivation and the state-of-the-art. Section 1.1 highlights the motivation behind the thesis. This is followed by Section 1.2 which presents the state-of-the-art countermeasures against power-based side-channel attacks on AES implementations. Section 1.3 covers the contribution of the thesis. Lastly, Section 1.4 presents the thesis outline.

1.1 Motivation

Increasing digitization in everyday aspects of modern life has resulted in today's technological world being highly data-driven. The data stored, processed and transmitted in this digital world is prone to vandalism, manipulation or theft during transmission, storage or computation. Encryption is a technique used to obfuscate information into a code, to limit or prevent unauthorized access by entities. It finds applications in critical economic sectors such as banking operations [1], access control [2], internet security [3] etc. Thus, information security is critical in the modern world. This has led to a need for security measures and regulations validating the security of a system handling data in the digital world.

This research focuses on the commonly used encryption technique known as Advanced Encryption System (AES). The algorithm is computationally secure, i.e, it cannot be broken with modern computer technology in a practical time period [4]. AES is the industry standard for symmetric-key encryption and finds implementations on both hardware and software in various sectors of the economy.

Physical implementations of encryption schemes on hardware and software is, however, found to be susceptible to side-channel attacks. Side-channel attacks require the adversary to capture and analyze sources of information-dependent leakage (power [**?**], time [5], noise [6], radiation [7] etc.). The attacks allow complete recovery of the encryption key with inexpensive resources. These techniques allow the adversary to circumvent the computational security of the encryption scheme and to decrypt the encrypted data within a practical time frame. Figure 1.1 provides an illustration of the steps involved in a typical side-channel attack. The security vulnerability posed by computing systems calls for a need to create countermeasures against side-channel attacks.

Power analysis has proven to be the most effective side-channel technique due to the ease of measurement and low sensitivity to noise [8]. Power consumption of a device is highly data-dependent and correlation patterns between power and data being operated upon may be obtained which can be used to break the encryption. Countermeasures against power analysis techniques are implemented on various abstraction levels in devices to increase the difficulty of breaking an encryption scheme. These countermeasures result in increased size, power consumption or area of the cryptographic implementation

at the cost of increased security [9]. Each countermeasure exhibits a different degree of success against various attacks and often involves a trade-off between practical implementation aspects such as cost, power, area, and security.

Power analysis countermeasures are hence vital in maintaining the security of the digital world. Thus, this thesis focuses on developing a novel countermeasure against power analysis techniques. Exploration of different power analysis attacks and countermeasures implemented at various abstraction levels is important to better understand the security and cost of the implementation. In the next section, the state-of-the-art countermeasures against power analysis techniques is covered.

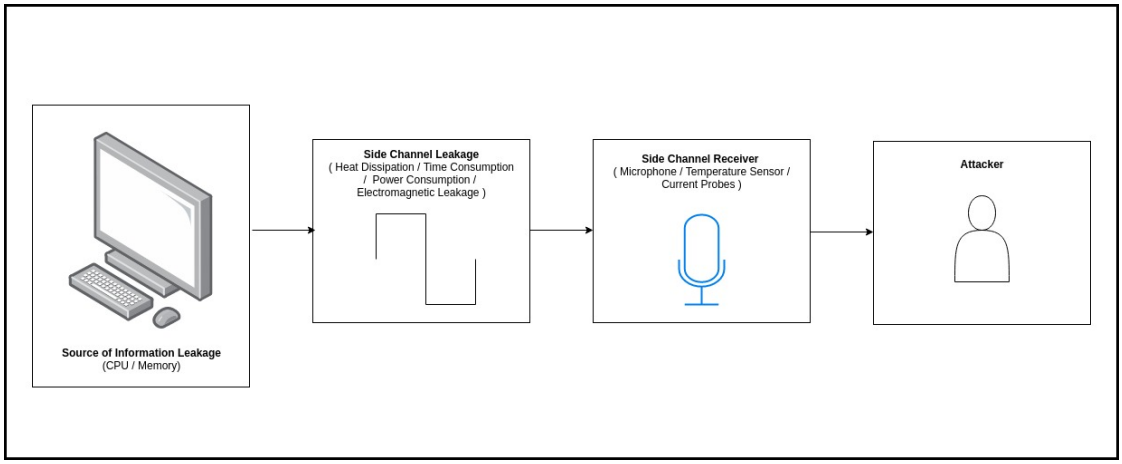

Figure 1.1: Side Channel Attacks

1.2 State-of-the-art Countermeasures

Power analysis countermeasures primarily aim at removing the possible correlation between the power consumed by the device and the secret key used in the encryption scheme. The security of the countermeasures are dependent on the attack being performed and no single countermeasure can effectively protect against all possible attacks. Thus, it is highly important to understand the system at different abstraction levels to perform a threat analysis of potential attacks in order to develop solutions against these attacks. Countermeasures are broadly classified into hiding and masking countermeasures.

Masking countermeasures are techniques that change the value of operands which are dependent on the secret key in order to conceal the computation. This is done through addition or multiplication of random variables to the intermediate values during the encryption process. The first known application of masking to counter power analysis techniques was proposed in 1999 [10], shortly after Paul Kocher [11] introduced power based side-channel attacks. In [9], this technique was found to be insecure against second-order power analysis techniques and was improved. Higher-order masking increased the area of the implementation exponentially for every order of attack it was designed to prevent. In [12], the authors proposed an efficient technique that allowed

conversion between boolean and multiplicative masking of intermediate values. This masking technique was proved to be insecure against zero value and collision attacks. A stronger countermeasure secure against these attacks was proposed in 2005 through a combination of additive and multiplicative masks [13]. Masking techniques have proven to be secure against power analysis techniques at great cost. Through the years, there have been attempts to optimize and modify the masking schemes in order to make them less time consuming, energy and area efficient. The authors in [14] concluded that a third-order masking scheme is 100 times slower than the original AES implementation.

Hiding countermeasures attempt to hide the computation without changing the value of operands which are dependent on the secret key. This is achieved by randomizing or smoothing the power consumed to make it constant during all operations. The first known application of this technique was published in 2002. In the proposed technique [15], the authors implement dual-rail logic style systems at the gate level in which each input and output signal have complementary signals, thus balancing bit-flips for every intermediate value. This technique while effective, results in doubling the area utilization and power consumed by the circuit. Multiple logic styles for counteracting power analysis attacks are released with a varied level of success against different attacks over the years. In 2008, the balancing was successfully implemented on architecture level [16] by implementing two parallel computing cores each operating on complementary data. Hiding countermeasures can also be implemented by the insertion of delay and dummy elements in the chain of operations. This creates a misalignment in the traces and reduces the effectiveness of an adversary's side-channel analysis on the implementation. Hiding countermeasures only moderately increases the number of traces that are required by an adversary to break the encryption and do not require any modification of the cryptographic algorithm.

The above passage clearly indicates that with the increasing sophistication of attacks, there is a need to constantly update existing countermeasures. This thesis work aims at analyzing and classifying existing attacks and countermeasures for AES implementations. It also proposes a novel countermeasure based on neural networks and tests the effectiveness of the countermeasure against power analysis attacks. The question is whether the power profile of a neural network is capable of reducing or removing the correlation between power consumption and data.

1.3 Contribution

In this thesis, the primary objective is to propose and test a neural network based countermeasure. The effectiveness of the countermeasure against different attacks is measured by measuring the confidence with which the encryption key can be extracted using different power based side-channel attacks. The main contributions of this thesis are:

- **Proposal of a novel class of countermeasures:**. This thesis proposes a novel type in addition to the existing solutions of hiding and masking techniques. It is categorized as "breaking". The linear power characteristics assumed by existing leakage models is broken resulting in secure implementation of the implementation

of the S-Net.

- **A conference submission on the proposal and implementation of a neural network based countermeasure (S-Net) in software:**. This thesis provides the design methodology of the S-Net in software. The security of the design is measured and validated using the Riscure experimental platform for side-channel analysis.

- **Measurement of security of implementation against Side Channel Attacks:** This thesis evaluates the security of the implemented countermeasure against Differential Power Analysis (DPA), Correlation Power Analysis (CPA), Multivariate and Deep Learning based template attacks. It is observed that the secret key of the unprotected implementation of AES can be extracted with less than 3000 power traces. Results indicate that the implemented countermeasure is secure against all attacks. Due to resource and time constrains, the number of traces has been limited to 20000.

- **Preparation of an experimental platform to perform security evaluations:** This thesis provides a description of the various devices, software and techniques involved in evaluating the security of the implemented countermeasure.

- **A classification and study of power based side-channel attacks and countermeasures**. This thesis classifies power based side channel attacks into profiled and non profiled techniques and provides an explanation of the steps involved in attacking protected and unprotected implementations of cryptographic algorithms using these techniques. In addition, it classifies existing countermeasures into hiding and masking techniques. This classifications gives a unique perspective on the security of protected implementations of encryption schemes.

1.4 Thesis Organization

The remainder of this thesis report is organized into five chapters. The first two chapters provide a background into topics that are relevant to the area of research. The next chapter explains the motivation and design methodology of the proposed countermeasure. This is followed by a description of the experimental platform used to validate the security of the countermeasure. This chapter also provides an evaluation of the security and the performance of the implemented countermeasure. The last chapter concludes the thesis. The following list provides a more detailed description of the topics discussed in each chapter:

Chapter 2 gives a brief background of the Advanced Encryption Standard (AES) algorithm. Thereafter, it illustrates in detail the power based side-channel analysis techniques on AES. This is followed by a brief literature review covering the design and working principle of known countermeasures.

Chapter 3 gives a brief introduction to Artificial Neural Networks. Thereafter, it illustrates in detail the different hyperparameters that affect the performance of the

neural network. This is followed by an explanation of the backpropagation algorithm and the steps involved in the training and inference phase of the neural network.

Chapter 4 first illustrates the motivation behind usage of neural networks as a countermeasure (S-Net). The chapter then presents the design methodology of the S-Net and optimization techniques used on the neural network architecture.

Chapter 5 first highlights the experimental platform used to validate the security of the countermeasure. It illustrates the implementation details of the S-Net. Thereafter, it provides an evaluation of the security of the implementation against various attacks. Lastly, it provides a performance analysis of the implemented countermeasure.

Chapter 6 concludes this thesis and presents future research directions.

AES Power Analysis Attacks and Countermeasures

2

AES (Advanced Encryption Standard) is a commonly used block cipher algorithm. It was initially proposed by Joan Daemen and Vincent Rijmen as the Rijndael cipher [17] and was selected by NIST as the new encryption standard in 2000 due to it's simplicity, security and ease of implementation. Once implemented, the algorithm was found to be vulnerable to power analysis attacks. The effectiveness of these attacks depend on it's complexity, the adversary's capabilities and the signal acquisition equipment. Secure implementations are thus required to alter the underlying algorithm or implementation as a countermeasure against these attacks. Section 2.1 covers the AES algorithm and highlights the steps involved in the encryption and decryption process. Section 2.2 provides a classification of different power analysis attacks on AES implementations. Section 2.3 highlights proposed countermeasures to power analysis attacks.

2.1 Advanced Encryption Standard

An in-depth understanding of the steps of the AES algorithm is beneficial in identifying the inherent weaknesses of the algorithm to power analysis attacks. AES is a symmetric encryption algorithm that is capable of processing 128 bit data blocks at a time. The key lengths can be 128, 192 or 256 bits. It is an iterative algorithm where a series of transformations are applied iteratively in each 'round'. The encryption process involves ten iterative rounds for 128-bit keys.

With the exception of the last round, each round involve an identical set of operations. A key is generated for each round from the input master key using a key scheduling (**KeyExpansion**) algorithm. The key scheduling process is highlighted in the subsection below. As shown in Figure 2.1, the encryption and decryption procedure consists of four transformations on the input data block. These include the addition of a round key (**AddRoundKey**), byte-wise substitution(**SubBytes**), a row-shift function (**ShiftRows**) and a mixing function of columns (**MixColumns**). The target encryption algorithm in this thesis is the 128-bit key AES, which has a total of 10 rounds. The high-level sequence of operations of the algorithm is shown in Figure 2.1. Each transformation is highlighted in detail in the following subsections.

AES operates on 4×4 column-major order matrix also known as the state matrix. Each column of the state matrix represents 32 bits and is also known as a word. Each round of encryption of AES performs the round operations on the input state matrix and produces a output state matrix to be fed to the next round. Figure 2.2 shows an example of the representation of the input plaintext, round key and ciphertext as matrixes.

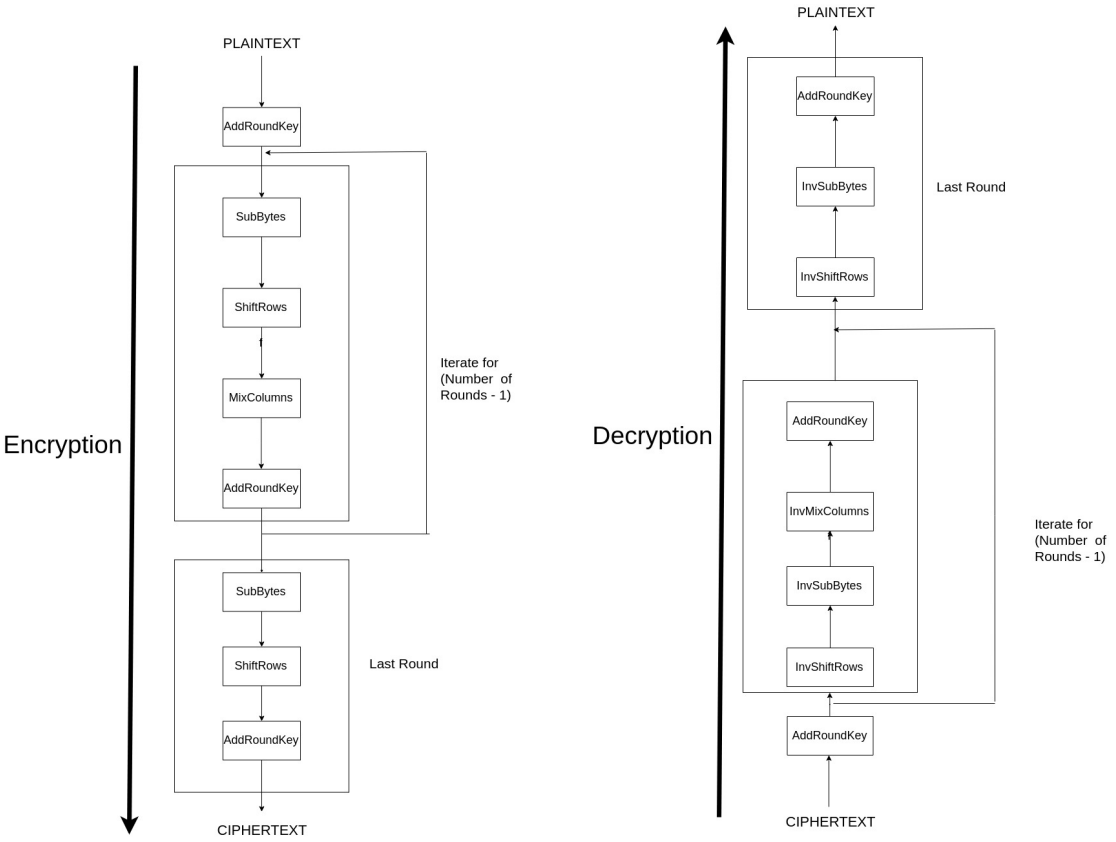

Figure 2.1: AES Encryption and Decryption

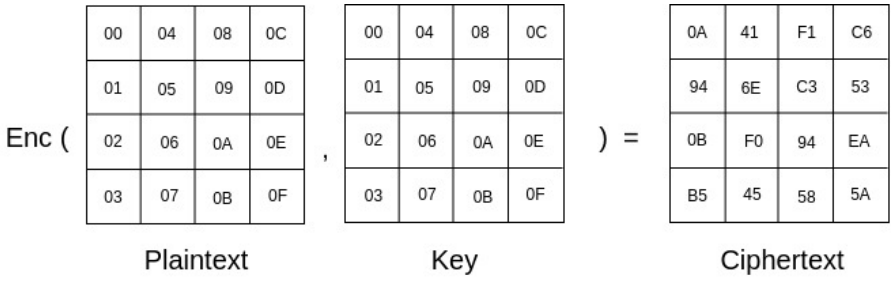

Figure 2.2: Representation of Input Plaintext, Key and Ciphertext

2.1.1 AddRoundKey

In this step, a byte-wise XOR is performed between the state matrix and the round key. The round keys are obtained as an output from the KeyExpansion step. The AddRoundKey function is called once before the encryption rounds and once in each encryption round.

2.1.2 SubByte

In this step, a byte-wise substitution function is applied on the state matrix. This results in $a_{i,j}$ being substituted substituted with $SubByte(a_{i,j})$ using a look-up table known as Substitution Box(S-Box). This is the point of attack for many side-channel attacks [13]. The table is either computed on-the-fly or stored in the memory depending on the area and memory constraints. The table is computed using two transformations which are highlighted below.

- The first step involves computation the multiplicative inverse of the input byte in $GF(2^8)$.
- This is followed by an affine transformation on the inverted byte over GF(2).

The Substitution box used in AES is illustrated in Fig 2.3. The input byte is used to select the row and column of the output byte. For example, an input byte "12" would point to "c9" as a result of the $SubByte$ operation.

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

Figure 2.3: Substitution Box used in AES

The S-Box is constructed to have good non-linearity properties [17]. It is also resistant to differential and linear cryptanalysis techniques [18]. In this manner, the correlation between input and the corresponding output is minimized. This minimized correlation paves the path for side-channel attacks on implementations of AES [19].

2.1.3 ShiftRows

This step involves the application of a linear transformation to each row in the state matrix. This is done by shifting each row in a cyclic manner to the left as illustrated in Fig 2.4. The goal of this step is to create a diffusion in the state bits.

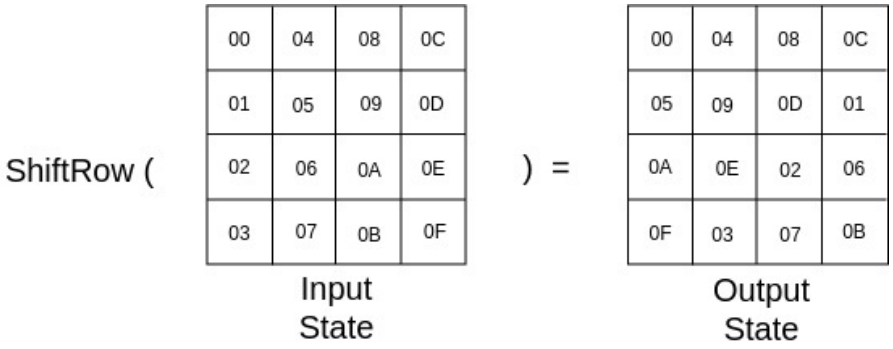

Figure 2.4: Shift Rows Operation used in AES

2.1.4 MixColumns

In this operation, a linear transformation is applied on each column of the state matrix. A matrix multiplication is performed in GF(2^8) as illustrated in Equation 2.1. This operation is skipped in the final round of the encryption algorithm.

$$\begin{bmatrix} Out_{1,n} \\ Out_{2,n} \\ Out_{3,n} \\ Out_{4,n} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} In_{1,n} \\ In_{2,n} \\ In_{3,n} \\ In_{4,n} \end{bmatrix} \tag{2.1}$$

2.1.5 KeyExpansion

A key scheduling algorithm is used to obtain a number of round keys from the master key. The 10 round variant of AES requires eleven 128 bit round keys to be generated for the encryption scheme. Thus, the round keys are a total of 1408 bits. The key scheduling algorithm is as follows.

- K_0, K_1, K_2 and K_3 represent the thirty-two bit word components of the 128 bit master key.
- $W_0, W_1, W_2 \cdots W_{43}$ are the thirty-two bit word components of the ten round keys.
- The round constants used in each round is defined as follows.

  $\text{rc}_i = [01\ 02\ 04\ 08\ 10\ 20\ 40\ 80\ 1B\ 36]$
  Select the value of rc_i depending on the round i and substitute below
  $\text{rcon}_i = [\text{rc}_i,\ 00_{16},\ 00_{16},\ 00_{16}]$

- The operation RWord involves a one-byte left circular shift on the input array.

$$\text{RWord}([b_0, b_1, b_2, b_3]) = [b_1, b_2, b_3, b_0]$$

- The operation SWord applies the Substitution Box on each byte of the input word.

$$\text{SWord}([b_0, b_1, b_2, b_3]) = [SubByte(b_0), SubByte(b_1), SubByte(b_2), SubByte(b_3)]$$

- Then, for i = 0, 1$\cdots$ 43, the following steps are used to compute the 32-bit components of the round keys.

$$W_i = \begin{cases} K_i & \text{if } i < 4 \\ W_{i-n} \oplus \text{SWord}(\text{RWord}(W_{i-1})) \oplus \text{RoundConstant}_{i/4} & \text{if } i \geq 4 \text{ and } i \equiv 0 (\text{mod } 4) \\ W_{i-n} \oplus W_{i-1} & \text{otherwise} \end{cases}$$

- In this fashion, the 44 words are generated and then combined to form 11 round keys used in each round. Note that the first four words are the same as the original master key which is applied on the input plaintext before entering the first round.

The algorithm has been found to be secure against various cryptanalytic techniques. Given the large key size, it also becomes impractical for the adversary to use a brute force attack. However, the properties of AES that make it secure against theoretical techniques allow real-world implementations of the algorithm to be easily attacked as will be explained in the next section.

2.2 Power-based Side Channel Attacks

Power analysis attacks seek to exploit the data dependence in the power consumed during an encryption operation. Transistors are the building blocks of CMOS-based logic designs. The power consumed by a device can be approximated as an aggregate of the power activities of these individual elements. Observed power consumption is partly dependent on the switching activity of the individual transistors [20]. It is clear that the power utilization of a cryptographic implementation is at least partially dependent on the data being processed. This forms the principle of power analysis attacks. The success of these attacks are highly dependent on the strengths of the adversary including signal acquisition equipment, countermeasures on the device under attack, processing, time and memory constraints.

Side-channel attacks have made it clear that power dissipation considerations are not only important from a device reliability perspective, but also a security point of view. Thus, it is important to have an understanding of the behaviour of the device at transistor level to truly appreciate and protect against side-channel attacks.

In order to explain the relationship of the data being processed and the power characteristics of CMOS technology, it is necessary to look at the building blocks of every device. A CMOS inverter is used to produce logic functions and is the primary component of all integrated circuits including Microprocessors and Field Programmable Gate Arrays. A CMOS inverter is a commonly employed logic design where the output is set to binary voltage levels based on the input signals. It is evident that a change in the input value causes current flow and the dynamic power behaviour of the inverter. Power analysis attacks track the power consumption observed during this transition and correlate it with the value of the secret key under operation. To be precise, if an adversary

can define a leakage model which correlates the observed power consumed to the value of secret key bit under operation, a side-channel attack is successful.

2.2.1 Taxonomy of Side-Channel Attacks

Substantial progress has been made in power analysis attacks over the last two decades, resulting in a perpetually growing list of countermeasures. This expansive list of attacks calls for a need of categorization and classification of the various techniques. Power Analysis techniques can be broadly classified into profiled and non-profiled attacks based on the control the adversary has over the device [21]. While non-profiled attacks assume limited control over the device and require some knowledge of the implementation, profiled attacks assume possession of a clone of the device and freedom to perform operations on input data and key of his choice. This is used to create a complete profile of the device which is then used to extract the secret key. Taxonomy of the possible attacks allows a structured overview into the capabilities of the adversary and simplify the task of implementing countermeasures against a set of attacks. Thus, an extensive taxonomy and summarizing of these attacks would be useful to any research that would benefit from the design of countermeasures. This is illustrated in Figure 2.5 and each attack is explained in detail in the following subsections.

2.2.2 Non-Profiled Attacks

Non-profiled attacks assume the ability to collect and analyze a limited number of side-channel traces for known inputs and a fixed and unknown key by the adversary [22]. The adversary does not possess the ability to perform encryptions using a key of his choice. Some working knowledge of the implementation is assumed. Examples of non-profiled attacks are highlighted in the section below.

2.2.2.1 Simple Power Analysis

Simple Power Analysis (SPA) [23] involves observation of the changes in the power consumed by the target device during execution. SPA attacks are carried out by visually inspecting the power trace and does not require any statistical analysis on the trace. It is helpful in identifying the areas of interest in iterative encryption algorithms. For example, one can easily observe ten rounds in the power consumption of unprotected AES. This is illustrated in Fig 2.6. Thus, SPA is extremely useful in locating important operations of an encryption scheme or identifying the encryption scheme being used.

In some implementations, it is also possible to extract the key being used during encryption. For example, the square and multiply algorithm is commonly used for exponentiation in an unprotected RSA implementation. This implementation can be successfully attacked using Simple Power Analysis [25]. The power peaks for the square and multiply operations are observably different and the key can easily be extracted. This is illustrated in Fig 2.7 [26]. Thus, if a specific instruction (or set of instructions) with observably different power characteristics are executed dependent on the key bits under operation, it is feasible to obtain the entire key by observing the power traces. The KeyExpansion step of AES has also proven to be insecure against SPA techniques.

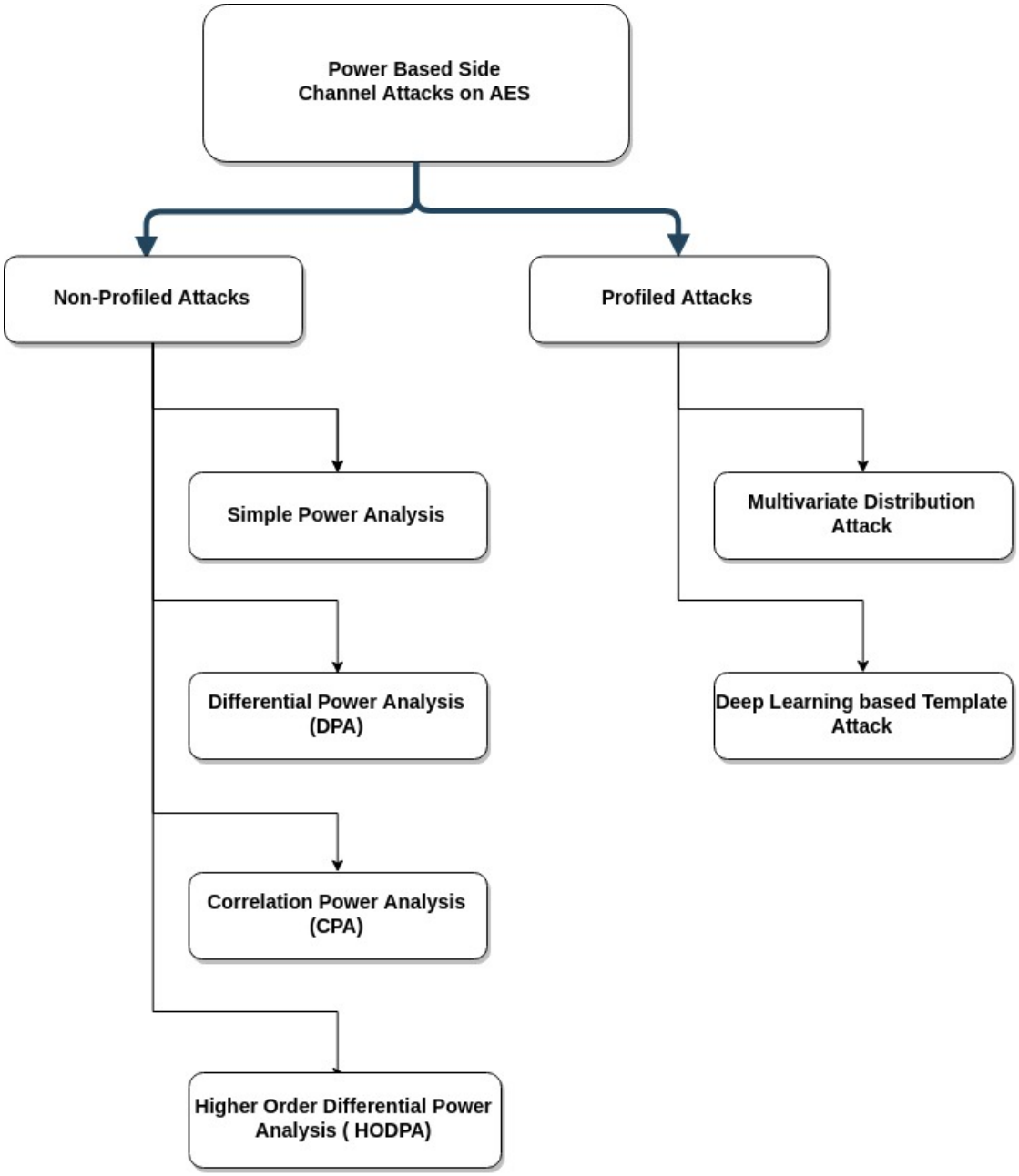

Figure 2.5: Taxonomy of Side Channel Attacks on AES

In [27], the proposed attack against AES implementations has proven to considerably reduce the key space to be searched by using brute force techniques.

2.2.2.2 Classic Differential Power Analysis (Difference of Means)

Differential Power Analysis involves the use of a hypothesis in order to extract the secret key from an encryption algorithm [8]. The power consumed is modeled based on a

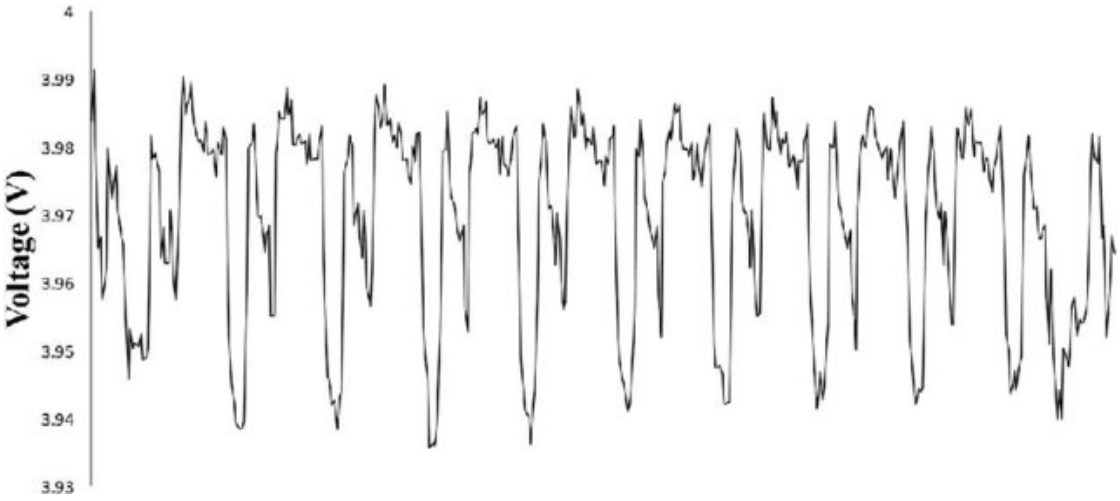

Figure 2.6: Identification of 10 rounds of AES Encryption using SPA [24]

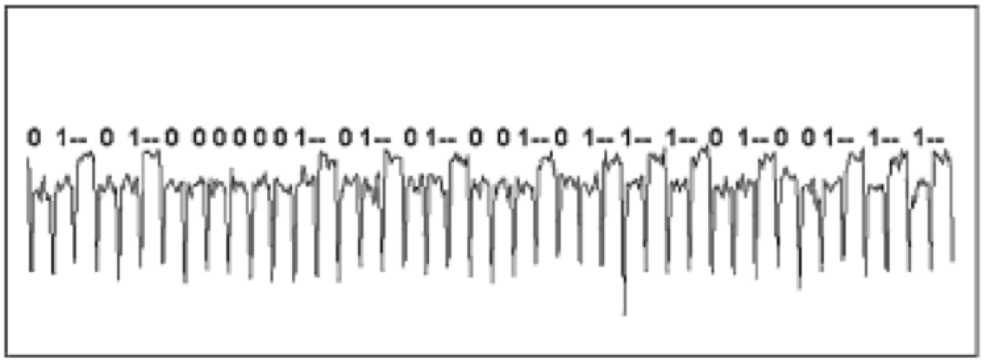

Figure 2.7: Leakage of Key Bits from an RSA implementation [26]

hypothesis on an operand which is linked to the secret key. This is followed by validation of the correctness of the hypothesis using the measured power traces. This is done by segregating the power traces into groups based on each bit of a calculated intermediate value. An observable difference between the average of the power traces between each group indicates correctness of the hypothesis. Thus, this model evaluates the correctness of a guess in a bitwise manner. The attack consists of the following steps:

1. **Select an Intermediate Value in the Algorithm to attack**. In this step, a point of attack in the targeted cryptographic algorithm is selected. In AES, the output from the *SubByte* operation is the result of a non-linear function applied on the input plaintext and the key. This non-linearity ensures that an incorrect hypothesis is easily identifiable in this attack. This makes it an ideal candidate for a selected point of attack.

2. **Acquisition of power traces**. In this step, power consumed by the device is

recorded in the form of power traces. Each trace corresponds to the encryption of a random chosen plaintext with a constant key. The corresponding known plaintext for each trace is also recorded along with each power traces. The recorded values is stored in a matrix, PT with dimension $t \times n$ as shown in Equation 2.2. The corresponding plaintext for each trace is stored in array $Plaintext$. Here, n represents the number of points in the trace and t represents the number of traces collected.

$$PT = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{t,1} & p_{t,2} & p_{t,3} & \cdots & p_{t,n} \end{bmatrix} \tag{2.2}$$

$$Plaintext = \begin{bmatrix} Plaintext_1 \\ Plaintext_2 \\ . \\ . \\ . \\ Plaintext_t \end{bmatrix} \tag{2.3}$$

3. **Formulation of hypothesis**. In this step, a hypothesis on a $SubKey$, h_k is formulated and the corresponding byte of the Substitution operation, $A_{x,k}$ is calculated for each trace (indicated by x) collected. The intermediate value is computed as the result of the $SubByte(Plaintext_x \; xor \; h_k)$ operation. This step and the following steps are iterated for 256 values of k. The calculated values can be stored in an array, A_k where each row corresponds to the intermediate value generated by each trace.

$$A_{x,k} = SubByte(Plaintext_x \; xor \; h_k) \tag{2.4}$$

$$A_k = \begin{bmatrix} A_{1,k} \\ A_{2,k} \\ . \\ . \\ . \\ A_{t,k} \end{bmatrix} \tag{2.5}$$

4. **Partition of the Traces**. In this step, the acquired traces in Step 2 are partitioned into two groups according to the value of one of the bits of the intermediate value,A_k obtained in Step 3. This step is iterated for every bit of the intermediate value and all possible values of k. For example, the partitions can be formed according the value of the LSB of the intermediate value for a given k. Here, $PT_{0,k}$ indicates those power traces where the LSB of the intermediate value is 0 and $PT_{1,k}$ indicates those power traces where the LSB of the intermediate value is 1.

$$PT_{0,k} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{t,1} & p_{t,2} & p_{t,3} & \cdots & p_{a,n} \end{bmatrix} \tag{2.6}$$

$$PT_{1,k} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{t,1} & p_{t,2} & p_{t,3} & \cdots & p_{b,n} \end{bmatrix} \tag{2.7}$$

5. **Evaluation**. In this step, the average of the power traces in each partition is computed and subtracted from each other. A correct key hypothesis, h_k would result in large peaks in the differential trace, which can be validated through inspection or automation. An incorrect hypothesis would result in a differential trace with smaller or no peaks. The position of the large peaks indicate the portions of the trace where the hypothesized bit is being used. Thus, this attack reveals both the correct $SubKey$ and the exact point of leakage. Here, $M_{0,k}$ indicates the average power consumption in $PT_{0,k}$ and $M_{1,k}$ indicates the average power consumption in $PT_{1,k}$. D_k indicates the difference between the average power consumption of the two groups.

$$M_{0,k} = \begin{bmatrix} m_{0,1} & m_{0,2} & m_{0,3} & ... & m_{0,n} \end{bmatrix} \tag{2.8}$$

$$M_{1,k} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & ... & m_{1,n} \end{bmatrix} \tag{2.9}$$

$$D_k = \begin{bmatrix} d_1 & d_2 & d_3 & ... & d_n \end{bmatrix} \tag{2.10}$$

6. **Result**. The array D_k is computed for all values of k. The array which contains the highest valued element among the D_k arrays corresponds to the correct hypothesis on the $SubKey$ h_k.

7. **Differential Coefficient**. The maximum value of D_k for each value of k is calculated and visualized.

The success of a DPA attack is dependent on the choice of the partitioning function. Partitioning functions can also be a function of multiple bits of the chosen intermediate value or a leakage model. Figure 2.9 and 2.8 shows graphs illustrating the variation in the difference of means for a correct and incorrect hypothesis on the $SubKey$ h_k [28]. With increase in the number of traces collected, the difference of the averages of the trace groups will converge to either the mean of the distribution or to zero depending on the correctness of the hypothesis. This method of DPA is mostly limited to evaluating one bit at a time and is time consuming. Stronger correlation models are used in more advanced versions of side-channel attacks as described in the following sections.

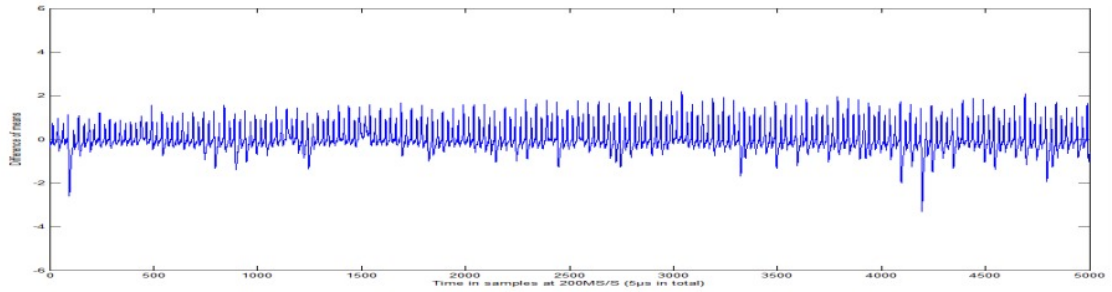

Figure 2.8: Differential trace for incorrect hypothesis on the $SubKey\ h_k$ [28]

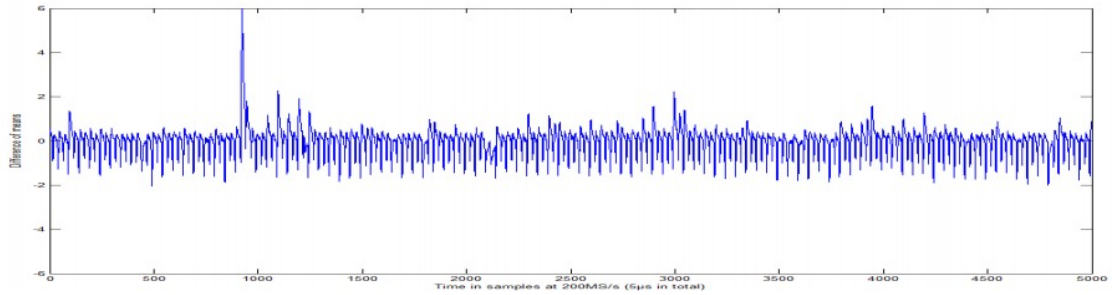

Figure 2.9: Differential trace for correct hypothesis on the $SubKey\ h_k$ [28]

2.2.2.3 Correlation based Differential Power Analysis

Correlation Power Analysis [29] is a powerful variant of the classical Differential Power Analysis. It involves the evaluation of the degree of correlation between a formulated leakage model and actual measurements. The chosen leakage model is applied on the hypothetical intermediate value and the correctness of the hypothesis affects the calculated correlation with the power traces. Some leakage models that are used for modelling hypothetical power consumption are explained below.

1. **Hamming Weight Model (HW)**. This model is a representation of the number of bits set to '1' in a chosen value. The range of this value varies from 0 to 8. This power model successfully captures the variation in power consumed by a transistor circuit holding a 'one' or a 'zero' [30]. An example is shown below.

$$\mathrm{HW}(0 \times 86) = \mathrm{HW}(1000\ 0110) = 4$$

2. **Hamming Distance Model (HD)**. This model is a representation of the distance between two values. This is done by counting the number of 0 to 1 transitions during an operation. This power model successfully captures the variation in power drawn by a transistor during transition of it's output value [30]. An example is shown below.

$$\text{HD}(0 \times 86 \rightarrow 0 \times 44) = \text{HD}(1000\ 0110 \rightarrow 0110\ 0110) = 3$$

3. **Zero Value Model (ZV)**. This model is a representation of the difference in power consumption when data being operated on is zero and non-zero [30]. An example is shown below.

$$\text{ZV}(0 \times 86) = 0$$
$$\text{ZV}(0 \times 00) = 1$$

The attack consists of the following steps:

1. **Select an Intermediate Value in the Algorithm to attack**. Similar to Step 1 of the classical DPA, the adversary selects an intermediate value to attack.

2. **Acquisition of power traces**. Similar to Step 2 of the classical DPA, the adversary records a set of power traces along with the plaintext under encryption.

$$PT = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{t,1} & p_{t,2} & p_{t,3} & \cdots & p_{t,n} \end{bmatrix} \tag{2.11}$$

$$Plaintext = \begin{bmatrix} Plaintext_1 \\ Plaintext_2 \\ . \\ . \\ . \\ Plaintext_t \end{bmatrix} \tag{2.12}$$

3. **Calculation of hypothetical Intermediate value**. Similar to Step 3 of the classical DPA, a hypothesis is made on the $SubKey$, h_k and the corresponding result of the $SubByte$ operation is calculated for each trace and corresponding plaintext, $Plaintext_x$. This step and the following steps are iterated for all possible hypotheses of the $SubKey$, i.e 256 values.

$$A_{x,k} = SubByte(Plaintext_x \ xor \ h_k) \tag{2.13}$$

$$A_k = \begin{bmatrix} A_{1,k} \\ A_{2,k} \\ . \\ . \\ . \\ A_{t,k} \end{bmatrix} \tag{2.14}$$

4. **Application of power leakage model**. In this step, a leakage model is applied to every element in the matrix A_k. The choice of leakage model is dependent on the target device. The mapped values are represented by array B_k. These arrays are generated for all values of k. The arrays are concatenated together to form a matrix of leakage values B.

$$B_k = \begin{bmatrix} l_{1,k} \\ l_{2,k} \\ . \\ . \\ . \\ l_{t,k} \end{bmatrix} \tag{2.15}$$

$$B = \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & ... & l_{1,255} \\ l_{2,1} & l_{2,2} & l_{2,3} & ... & l_{2,255} \\ . & . & . & . & . \\ . & . & . & . & . \\ l_{t,1} & l_{t,2} & l_{t,3} & ... & l_{t,255} \end{bmatrix} \tag{2.16}$$

5. **Evaluation**. In this step, the degree of correlation between the matrix B and the acquired power traces in PT is calculated. This is calculated by substituting the appropriate values in Equation 2.17. The results are stored in matrix C.

$$c_{i,j} = \frac{\sum^n (l_{n,i} - \mu_{l_i})(p_{n,j} - \mu_{p_j})}{\sqrt{\sum^n (l_{n,i} - \mu_{l_i})^2 (p_{n,j} - \mu_{p_j})^2}} \tag{2.17}$$

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & ... & c_{1,n} \\ c_{2,1} & c_{2,2} & c_{2,3} & ... & c_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ c_{255,1} & c_{255,2} & c_{255,3} & ... & c_{255,n} \end{bmatrix} \tag{2.18}$$

6. **Result**. The row which corresponds to the highest valued element among each column of the matrix elements corresponds to the correct hypothesis on the $SubKey \ h_k$. The location where the $SubKey$ is used in the encryption operation indicated by n.

2.2.2.4 Higher Order Differential Power Analysis

Higher Order DPA (HODPA) is a powerful variant of the differential power analysis which attacks multiple intermediate values simultaneously [31]. The attack combines multiple samples within a trace that corresponds to multiple intermediate values using a combination function.

A common application of HODPA is used when the target intermediate value x is masked with a random mask m. If a second known intermediate masked variable, y can be found with the same mask m, the values can be XORed with each other to obtain the unmasked sum of the variables. An hypothesis can be formulated on this combination and correlated with the power traces as shown in the classical DPA attacks. For example, in a masked AES implementation, the input to the Substitution box, x is normally masked with mask m, to obtain x_m. The output of the Substitution box, y_m contains the same mask. A HODPA attack can be performed on this implementation by creating a hypothesis on $x_m \oplus y_m$, which is equal to $x \oplus y$ which can be predicted [32].

2.2.3 Profiled Attacks

Profiled attacks assume the possession of a clone of the target device capable of encrypting a chosen plaintext with a chosen key. The recorded power traces obtained from this clone during encryption is used to construct a template. This template is then compared with the behaviour of the target device during encryption to determine the key used in the encryption scheme [33]. Profiled attacks are powerful and strong templates allow the adversary to extract the hidden key with a single power trace.

Profiled attacks usually comprise of two phases: the Profiling and Extraction phase. The Profiling phase involves the adversary capturing the behaviour of the clone which is in turn used to create a template. The steps involved in the creation of the template is dependent on the attack model used. This section highlights the steps involved in two attack models i.e Multivariate distribution based template attack and the Deep learning based template attacks.

2.2.3.1 Multivariate Distribution Template Attacks

The attack approximates the leakage observed in the power traces as a multivariable distribution, represented using a covariance matrix and a mean vector. The adversary computes the two components for each key and input and uses this information to create a template. This template is then used to analyze the target trace to recover the secret key with the highest probability of being used [34].

This profiling phase for this attack consists of the following steps:

1. **Procurement of cloned device**. In this step, the adversary procures a clone of the device to be attacked.

2. **Acquisition of traces**. In this step, a set of power traces are recorded. Each trace corresponds to the encryption of a randomly chosen plaintext with a random key. The plaintext and key used for each trace is also recorded along with each power trace. The recorded values can be stored in a matrix with dimension $t \times n$

as shown in Equation 2.30. Here, n represents the number of points in the trace and t represents the number of traces collected.

$$PT = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{t,1} & p_{t,2} & p_{t,3} & \cdots & p_{t,n} \end{bmatrix} \tag{2.19}$$

3. **Processing of traces**. Full knowledge of the key used in the encryption of each trace is assumed in the Profiling phase. Thus, for each trace collected in the previous step, the adversary can compute the intermediate value. Similar to Step 4 of the previous attack, each output of the operation during encryption is mapped according to the chosen leakage model. The mapped values are stored in matrix B.

$$B = \begin{bmatrix} l_1 \\ l_2 \\ . \\ . \\ . \\ l_t \end{bmatrix} \tag{2.20}$$

4. **Separation of Power Traces**. In this step, the adversary divides the traces collected based on the value of the mapped leakage model in matrix B. For example, in the Hamming Weight (HW) leakage model, the traces can be divided into nine possible groups.

5. **Selection of Points of Interest**. The adversary identifies m points of interest in the power trace. The points in traces that give the highest difference in power consumption for different Hamming Weights are selected as points which are indicative of the key used. This is done by computing the mean vector of the power trace for each Hamming Weight group and observing the difference between each vector. The points on the resulting power trace which indicate maximum difference are selected as the points of interest. The rest of the points in the power trace are not considered in the further steps.

6. **Creation of Template** After separation of the traces into groups and identification of the points of interest, the covariance matrix and the mean vector is computed for the traces in each group, indexed by i. The results are stored in matrix CV_i and array M_i. Thus, this step results in nine covariance matrixes and mean vectors if the Hamming Weight model is used.

$$CV_i = \begin{bmatrix} cv_{1,1} & cv_{1,2} & cv_{1,3} & ... & cv_{1,m} \\ cv_{2,1} & cv_{2,2} & cv_{2,3} & ... & cv_{2,m} \\ . & . & . & . & . \\ . & . & . & . & . \\ cv_{t,1} & cv_{t,2} & cv_{t,3} & ... & cv_{t,m} \end{bmatrix} \tag{2.21}$$

$$M_i = \begin{bmatrix} m_1 & m_2 & m_3 & ... & m_m \end{bmatrix} \tag{2.22}$$

The Extraction phase for this attack is highlighted below.

1. **Procurement of target device**. In this step, the adversary procures the target device to be attacked.

2. **Acquisition of traces**. Similar to Step 2 of the profiling phase, the adversary records multiple traces during encryption of random plaintext. Unlike the profiling phase, the adversary has no knowledge of the key used in the encryption. The recorded values can be stored in a matrix with dimension $t \times n$ as shown in Equation 2.32. Here, n represents the number of points in the trace and t represents the number of traces collected. In the extraction phase, the number of traces collected may be small.

$$PT = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{t,1} & p_{t,2} & p_{t,3} & \cdots & p_{t,n} \end{bmatrix} \tag{2.23}$$

3. **Application of Probability Density Function on target traces** Using the covariance matrixes, CV_i and the mean vectors, M_i that were computed in the previous step, the probability density function on each target trace, t is calculated for each template. Thus, given a set of templates that are characterized by (M_i, CV_i), on application of the probability density function on the traces, the resulting value indicates the probability that the particular leakage model value, i was used in the trace.

$$f_i(t) = \frac{1}{\sqrt{2\pi \times det(CV_i)}} \times exp((t - M_i)' \times CV_i^{-1} \times (t - M_i)) \tag{2.24}$$

The results of the computation is stored in matrix F. Each column indicates the value of the leakage model and each row represents a power trace. The Hamming weight is selected in the following equation. Thus the matrix would contain nine columns, one for each value of the leakage model.

$$F = \begin{bmatrix} f_{t_1,(M_0,CV_0)} & f_{t_1,(M_1,CV_1)} & f_{t,(M_2,CV_2)} & \cdots & f_{t_1,(M_8,CV_8)} \\ f_{t_2,(M_0,CV_0)} & f_{t_2,(M_1,CV_1)} & f_{t_2,(M_2,CV_2)} & \cdots & f_{t_2,(M_8,CV_8)} \\ . & . & . & . & . \\ . & . & . & . & . \\ f_{t_n,(M_0,CV_0)} & f_{t_n,(M_1,CV_1)} & f_{t_n,(M_2,CV_2)} & \cdots & f_{t_n,(M_8,CV_8)} \end{bmatrix} \tag{2.25}$$

4. **Application of power leakage model on intermediate value**. Similar to Step 3 and Step 4 of Correlation DPA, the adversary makes an hyptothesis on the 256 possible values of the Subkeys and calculates the intermediate value corresponding to each trace and plaintext. This information is stored in matrix A. Subsequently, a leakage model is applied on each element of A and the mapped values are stored in B.

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & ... & A_{1,255} \\ A_{2,1} & A_{2,2} & A_{2,3} & ... & A_{2,255} \\ . & . & . & . & . \\ . & . & . & . & . \\ A_{t,1} & A_{t,2} & A_{t,3} & ... & A_{t,255} \end{bmatrix} \tag{2.26}$$

$$B = \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & ... & l_{1,255} \\ l_{2,1} & l_{2,2} & l_{2,3} & ... & l_{2,255} \\ . & . & . & . & . \\ . & . & . & . & . \\ l_{t,1} & l_{t,2} & l_{t,3} & ... & l_{t,255} \end{bmatrix} \tag{2.27}$$

5. **Application of Probability Density Function on leakage model**. In this step, the elements of the matrix B is correlated with the with the elements of matrix F. We refer to this new matrix as C in which each element indicates the probability that the hypothesized Subkey was used in the power trace. The logarithmic sum of each column in C is calculated to produce an array S. The index of the element with the largest sum in S indicates the SubKey used in the power trace.

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & ... & c_{1,255} \\ c_{2,1} & c_{2,2} & c_{2,3} & ... & c_{2,255} \\ . & . & . & . & . \\ . & . & . & . & . \\ c_{t,1} & c_{t,2} & c_{t,3} & ... & c_{t,255} \end{bmatrix} \tag{2.28}$$

$$S = \begin{bmatrix} s_1 & s_2 & s_3 & ... & s_{255} \end{bmatrix} \tag{2.29}$$

2.2.3.2 Deep Learning Based Template Attacks

The Deep Learning Based Template Attack is similar to the Multivariate Distribution template Attack. The characterization of the template is however executed with the help of a trained deep learning neural network. The profiling phase for this attack consists of the following steps.

1. **Procurement of cloned device**. In this step, the adversary procures a clone of the device to be attacked.

2. **Acquisition of traces**. In this step, a set of power traces are recorded. Each trace corresponds to the encryption of a randomly chosen plaintext with a random key. The plaintext and key used for each trace is also recorded along with each power trace. The recorded values can be stored in a matrix with dimension $t \times n$ as shown in Equation 2.30. Here, n represents the number of points in the trace and t represents the number of traces collected.

$$PT = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{t,1} & p_{t,2} & p_{t,3} & \cdots & p_{t,n} \end{bmatrix} \tag{2.30}$$

3. **Characterization using Deep Learning Network**. The acquired traces are characterized by using a deep learning network. The labels used to train the deep learning network is the value returned by the leakage model for each trace. Since the adversary has information about the plaintext and key used in each trace, the corresponding intermediate value can be computed for each trace. The leakage model is then applied on the intermediate value to to be used as a label for each trace. The entire trace is passed as input to the neural network along with the corresponding label. Sixteen such neural networks are trained simultaneously for each subbyte of the key.

The Extraction phase for this attack is highlighted below.

1. **Procurement of target device**. In this step, the adversary procures the target device to be attacked.

2. **Acquisition of traces**. Similar to Step 2 of the profiling phase, the adversary records multiple traces during encryption of random plaintext. Unlike the profiling phase, the adversary has no knowledge of the key used in the encryption. The recorded values can be stored in a matrix with dimension $t \times n$ as shown in Equation 2.32. Here, n represents the number of points in the trace and t represents the number of traces collected. In the extraction phase, the number of traces collected may be small.

$$PT = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,n} \\ . & . & . & . & . \\ . & . & . & . & . \\ p_{t,1} & p_{t,2} & p_{t,3} & \cdots & p_{t,n} \end{bmatrix} \tag{2.31}$$

3. **Application of traces on Neural Network**. The traces collected are applied on the trained neural network. The probability of the label used is returned as the inference of the neural network. This is represented in Equation 2.32. Each value of the matrix represents the probability that leakage model value i is used in trace

t_j. For example, on usage of the Hamming Weight model as a label to train the neural network, the matrix would contain nine columns.

$$F = \begin{bmatrix} DL(p_1)_1 & DL(p_1)_2 & DL(p_1)_3 & ... & DL(p_1)_j \\ DL(p_2)_1 & DL(p_2)_2 & DL(p_2)_3 & ... & DL(p_2)_j \\ . & . & . & . & . \\ . & . & . & . & . \\ DL(p_t)_1 & DL(p_t)_2 & DL(p_t)_3 & ... & DL(p_t)_j \end{bmatrix} \tag{2.32}$$

4. **Application of power leakage model on intermediate value**. Similar to Step 3 and Step 4 of Correlation DPA, the adversary makes an hyptothesis on the 256 possible values of the Subkeys and calculates the intermediate value corresponding to each trace and plaintext. This information is stored in matrix A. Subsequently, a leakage model is applied on each element of A and the mapped values are stored in B.

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & ... & A_{1,255} \\ A_{2,1} & A_{2,2} & A_{2,3} & ... & A_{2,255} \\ . & . & . & . & . \\ . & . & . & . & . \\ A_{t,1} & A_{t,2} & A_{t,3} & ... & A_{t,255} \end{bmatrix} \tag{2.33}$$

$$B = \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & ... & l_{1,255} \\ l_{2,1} & l_{2,2} & l_{2,3} & ... & l_{2,255} \\ . & . & . & . & . \\ . & . & . & . & . \\ l_{t,1} & l_{t,2} & l_{t,3} & ... & l_{t,255} \end{bmatrix} \tag{2.34}$$

5. **Correlation of Hypothesis Matrix with Leakage Model**. Each element of matrix B is correlated with the elements of matrix F. This new matrix is referred to as C in which each element indicates the probability that the hypothesized Subkey was used in the power trace. The logarithmic sum of each column in C is calculated to produce an array S. The index of the element with the largest sum in S indicates the SubKey used in the power trace.

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & ... & c_{1,255} \\ c_{2,1} & c_{2,2} & c_{2,3} & ... & c_{2,255} \\ . & . & . & . & . \\ . & . & . & . & . \\ c_{t,1} & c_{t,2} & c_{t,3} & ... & c_{t,255} \end{bmatrix} \tag{2.35}$$

$$S = \begin{bmatrix} s_1 & s_2 & s_3 & ... & s_{255} \end{bmatrix} \tag{2.36}$$

2.3 Countermeasures

The Substitution Box operation is a primary point of attack in the AES algorithm, as seen in the previous section. Several countermeasures have been developed around this operation with different levels of security. These typically involve protection at the algorithmic level, and are independent of the device. Masking countermeasures typically involve combination of the intermediate values with a random value during computation in order to create random intermediates [35]. These random intermediates remove possible correlations between the power profile and the data being processed. There also exist countermeasures that are independent from the executed algorithm. For example, a register holding the intermediate value of an encryption operation can be duplicated to hold the complement of the intermediate value. This countermeasure is also known as the dual-rail scheme and belongs to the hiding class of countermeasures [36]. These countermeasures make no modification to the intermediate values generated during encryption. While there is no countermeasure which makes the algorithm completely secure against all possible attacks, the objective of the countermeasure aims at increasing the difficulty of the attack from a polynomial-time bounded adversary. Thus, it is clear that the ease of attacking an implementation of a cryptographic algorithm using side-channel attacks gives rise to the need for developing countermeasures against the same. This section highlights a classification, in-depth analysis of some existing countermeasures and their implementation.

2.3.1 Classification of Countermeasures

Countermeasures can be classified into two major categories based on their implementation. Hiding countermeasures aim at hiding the data being processed while the masking countermeasures are designed to apply a randomized mask on the data being processed. While masking countermeasures are strictly at an algorithmic level, hiding countermeasures can be classified further depending on the abstraction layer it is implemented in. This classification is illustrated in Figure 2.10. The different types of countermeasures are explained in detail in the following sections.

2.3.2 Hiding

The hiding countermeasure primarily aims at hiding the intermediate value with no modification to the value itself. This can be achieved by adding data-independent noise to the power consumption of the device. For example, addition of dummy instructions or shuffling of operations is an inexpensive and commonly used countermeasure [37]. Misalignment of traces causes the intermediate value to be processed at different points in each acquired power traces and can reduce the effectiveness of an attack [38]. This countermeasure however offers limited security as simple pre-processing techniques on the traces can remove the security of hiding techniques.

A hiding countermeasure can also be achieved by ensuring a constant power supply irrespective of the data being operated on. For example, pre-charged logic cells and balanced structures which process both the operation and it's inverse simultaneously to exhibit a constant power signature are commonly used countermeasures [39]. This

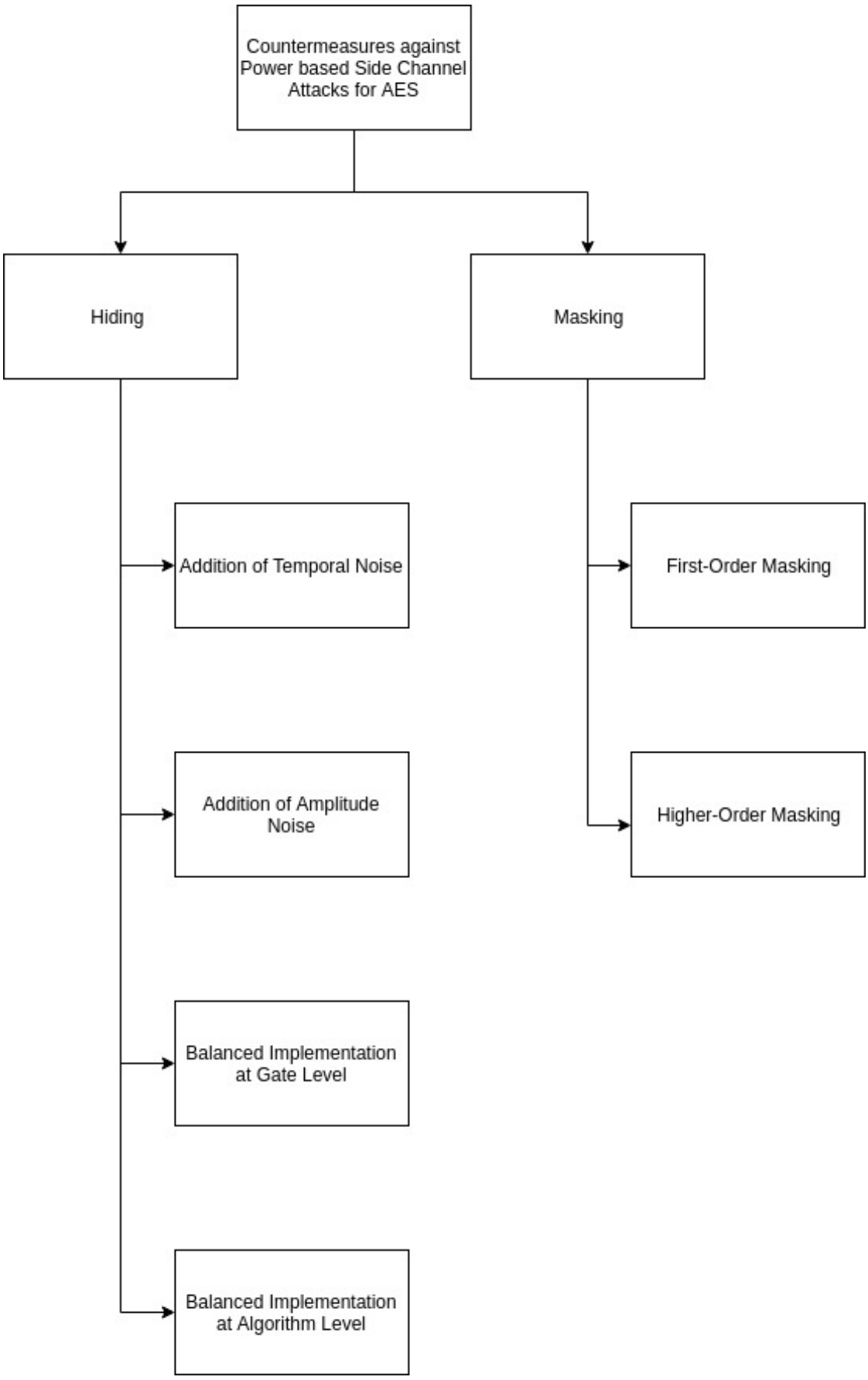

Figure 2.10: Classification of Countermeasures

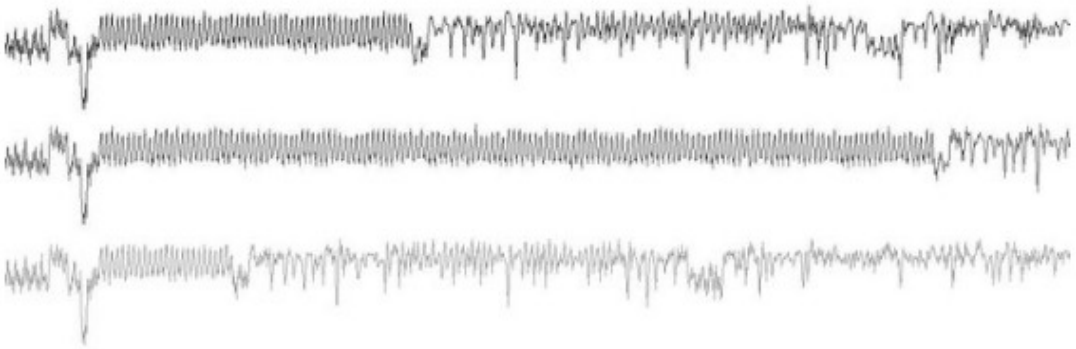

Figure 2.11: Random Delay as Countermeasure for AES [40]

research study highlights some commonly used hiding implementations in the following subsections.

2.3.2.1 Addition of Temporal Noise

Misalignment of traces through the addition of temporal noise causes the critical intermediate value to be processed at different points in each acquired power trace and can reduce the effectiveness of an attack. This is illustrated in Figure 2.11. Random insertion of instruction and delay modules are inexpensive and do not significantly add to the cost of implementation in terms of area, power and time. They can easily be added to both hardware and software implementations of the algorithm. A secure pseudo random generator must be implemented to calculate appropriate delay values [41]. For example, the number and choice of dummy instructions or delay elements to be added in the algorithm chain must not be a predictable pattern that can be easily 'edited out' from a power trace. Ambrose et al. proposed insertion of dummy instructions in the critical areas of the algorithm which read and write random values to registers when critical instructions are executed [42]. This is effective in disguising SPA patterns and makes the acquired traces harder to pre-process for DPA. This scheme also offers protection against higher-order attacks since alignment becomes more difficult when multiple points on the trace needs to be taken into consideration [43].

However, these protection mechanisms are vulnerable to processing techniques which may used by the adversary if the countermeasure is incorrectly implemented. Some techniques include transforming the traces acquired using Fourier transform and removing the effect of misalignment [44]. Insufficient randomness of the function dictating the position and number of the dummy operations inserted may also allow the adversary to simply remove the unwanted portions of the trace and proceed to mount a successful attack. In this case, delay elements can be easily observed using a Simple Power Analysis attack allowing the adversary to mount an attack on the relevant segments of the acquired power trace.

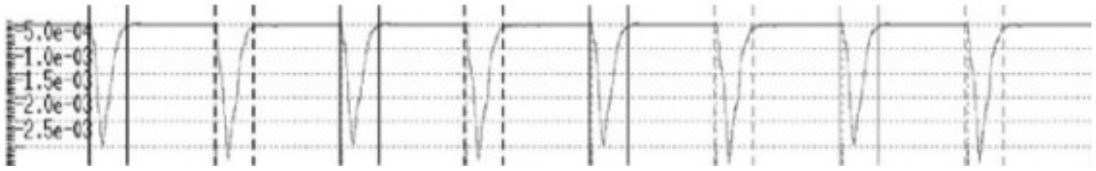

Figure 2.12: Unprotected Implementation of AES [45]

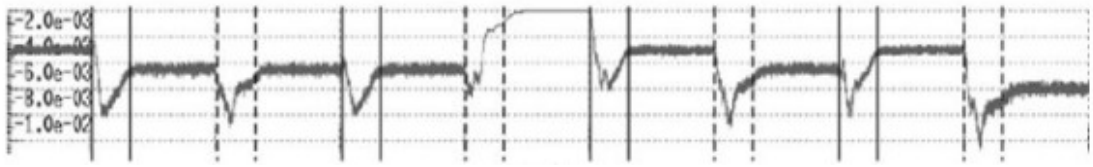

Figure 2.13: Protected implementation of AES using random amplitude noise [45]

2.3.2.2 Addition of Amplitude Noise

The number of traces required to obtain the key used would increase by decreasing amplitude of the signal with respect to the noise. This implementation makes it difficult for the adversary to observe data dependent variations [45] by hiding the leakage. This is illustrated in Figure 2.12 and 2.13. While these countermeasures are cheap and require minimal investment in terms of area, noise and power, their security is measured by the signal-to-noise ratio achieved and the attack used. It is important that the frequency response of the noise source must match with the signal. Incorrect implementation would allow the the adversary to pre-process the traces using a filter and remove the added noise. Some sources of amplitude noise includes noise generators [46] and random oscillators [47].

2.3.2.3 Balanced Implementation at Gate Level

Leakage models in side channel attacks rely on the detection of bit-flips or the value of the bits stored in the registers. This hiding countermeasure involves the flattening of the power consumed by the component making it independent of the bits involved in the operation. This is achieved by creating logic cells that process data along with it's inverse thus equalizing power consumption during execution of each instruction. Thus, each input and output signal is represented by complementary wires. The simultaneous charging and discharging of these complementary wires results in fixed transition counts which are data-independent. This breaks the leakage model and renders attacks ineffective. A notable example of this logic style is the Wave Dynamic Differential Logic (WDDL). Figure 2.14 illustrates basic logic gates designed using the WDDL logic style [48]. Shrinking transistor and wiring sizes results in implementation constraints making this countermeasure expensive to design correctly. For example, in case of WDDL, the loading capacitance between the two complementary wires must be the same. If designed incorrectly, the waveforms of each output signal can be distinguished and the attacks proves to be successful. A difference in delay time between each output signal may also al-

low the implementation to be attacked. Overheads include increased area along with increased power consumption in comparison to the unprotected design. Thus, usage of this countermeasure significantly increases design costs when implemented on a product.

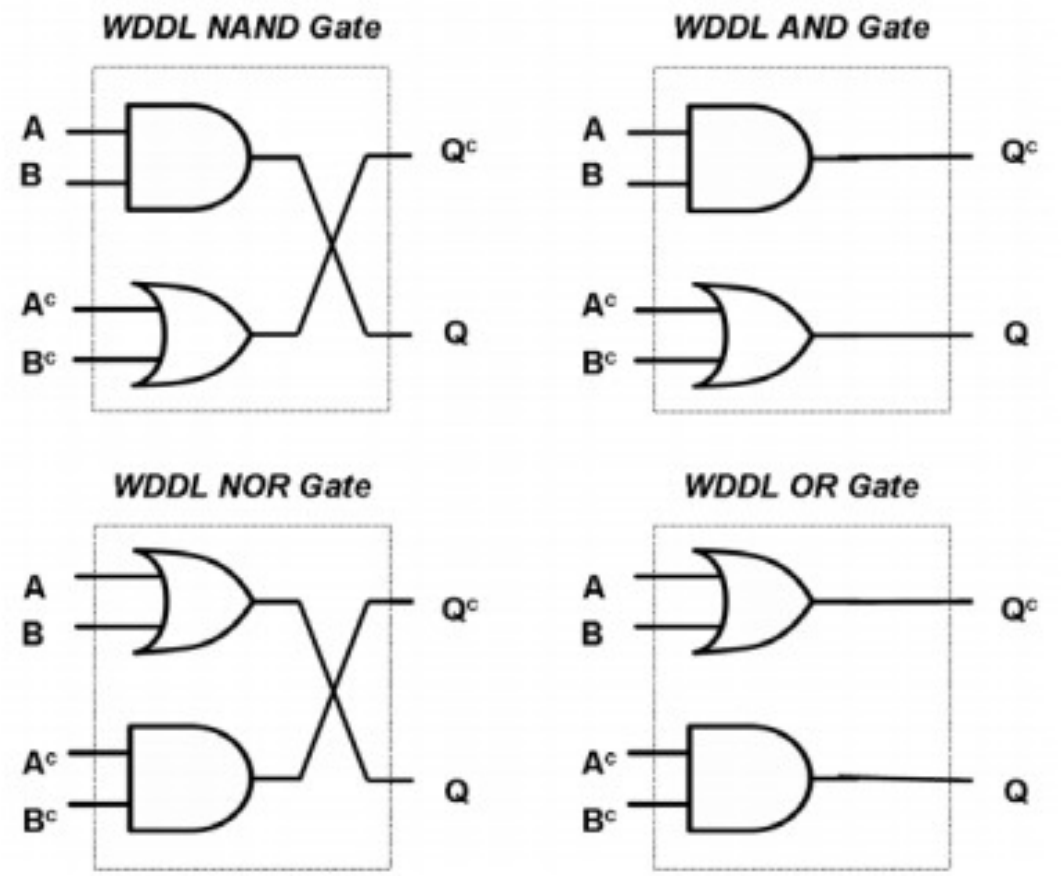

Figure 2.14: Design of logic gates using WDDL logic style [48]

2.3.2.4 Balanced Implementation at Algorithm Level

The implementation of a countermeasure using balanced logic, as highlighted in the previous section, is expensive in terms of area and power. A balanced design at a gate level abstraction would perform the balancing for all data processed through the gates. It would also require compilation of special libraries required to implement balanced logic. This section highlights a proposed countermeasure which performs the balancing at an architecture level. This technique makes use of a dual-processor architecture, where the two processors encrypt complementary data, thus balancing the power observed due to bitflips [16]. The second processor may be used only when data is being encrypted and can execute other tasks when encryption is not required by the application. Thus, this countermeasure uses twice the hardware only when performing secure operations.

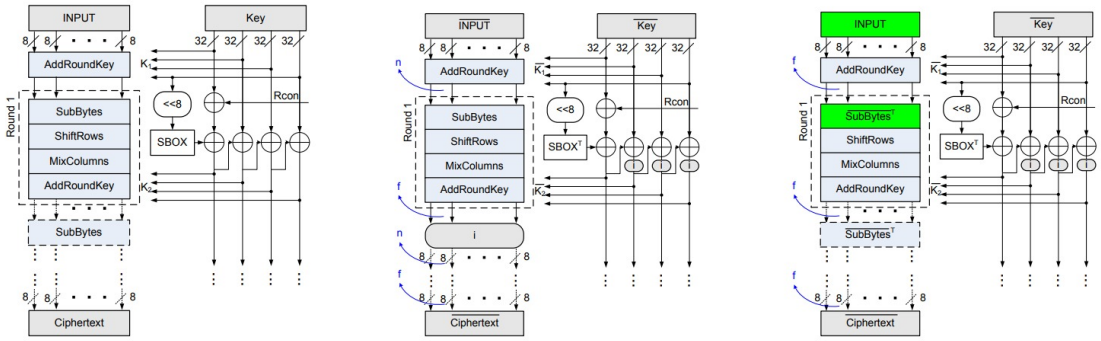

Figure 2.15: Balancing of AES Algorithm [16]

In order to obtain a complete inversion of the intermediate values processed during encryption, the second processor uses an inverted key and an inverted and transposed version of the *SubByte* operation, indicated by $\overline{SubByte^T}$. The S-Box used in the key-scheduling process is also transposed. Figure 2.15 depicts the implementation of the countermeasure in a step-by-step manner. The partial inversion is shown to indicate the effect of complementing the *Input* and the *Key* with a transposed S-Box. This partial inversion results in an inverted ciphertext. While this partial inversion produces a final inverted output, the intermediate data after the first *SubByte* remains the same as the original implementation making the implementation more vulnerable to power analysis attacks. This partial inversion implementation is further modified by keeping the *Input* unchanged and performing $\overline{SubByte^T}$. This results in retrieval of complementary values during the *SubByte* operation in comparison to the original implementation. Thus, the final design produces a complete inversion of all intermediate values obtained during the encryption and the power consumption is balanced making it secure against power analysis techniques. This implementation is secure against the Hamming Weight and Distance models for every order of DPA.

The place and route of the chip containing these processors must be performed such that it is not possible to isolate the power profile of one particular core. One possible threat model would be placement of an EM probe in the vicinity of a single core, which would render the balancing technique useless. The processors must also be completely synchronized with each other using an external synchronizer to ensure that each intermediate value (and it's inversion) are processed at the same time. An incorrect synchronizer allows the adversary to easily isolate the power trace of one of the processors.

2.3.3 Masking

The masking countermeasure primarily aims at concealing the critical intermediate value by applying a random mask to it. This is achieved by addition or multiplication of random values to the encryption input and intermediate values [49]. These 'masks' obfuscate

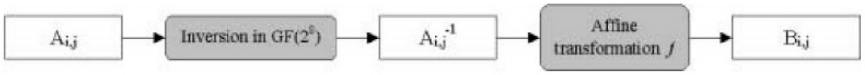

Figure 2.16: Unprotected SubByte operation [12]

the computation on actual values making the power trace uncorrelated to the secret key being used. The power consumption is made independent at an algorithmic level without making modification to the device characteristics. The masked representations of the intermediate values are computed and stored until the final round of the AES encryption algorithm. The original values are reconstituted from the final output of the final round. Masking schemes can be sub-classified into two categories based on the type of mask used to randomize the critical value, namely arithmetic and boolean masking. Arithmetic masking performs modular addition or multiplication on the chosen intermediate value. This masking scheme is suitable to mask non-linear functions such as SubByte. Boolean masks performs a bit-wise exclusive-or operation on the chosen intermediate value. This technique is used to mask intermediate values that are inputs to linear functions such as ShiftRows, MixColumns and AddRoundKey. The following sections highlight different masking techniques commonly deployed as a side-channel countermeasure.

2.3.3.1 Masking the Substitution Box against DPA

Non linear functions such as *SubByte* can be modified to allow masked inputs which results in masked outputs. For example, boolean masks are applied to the input of Substitution boxes as shown below [12]. Given the original SubByte operation S and input i to which the mask m_0 is applied, an appropriate output mask m_1 and new SubByte operation S' must be constructed in the following manner in order to recover the final output.

$$S'[i] = S[i \oplus m_0] \oplus m_1 \tag{2.37}$$

An alternate approach to masking is through decomposition of the SubByte operation into a series of linear and non linear operations [12]. The SubByte operation is a combination of a multiplicative inversion in GF(2^8) followed by an affine transformation, f applied on the input byte as illustrated in Figure 2.16. These operations can be individually masked as illustrated on a high level in Figure 2.17. An illustration of the modified inversion in detail is shown in Figure 2.18. The output $X1_{i,j}$ is computed as $f(X_{i,j}) \oplus 0 \times 63$ (derived from the affine properties of the transformation). The mask $X1_{i,j}$ at the end of the modified inversion is then transformed by the linear ShiftRow, MixColumns and AddRoundKey step until it finally removed at the end of the round with an additive exclusive-or addition.

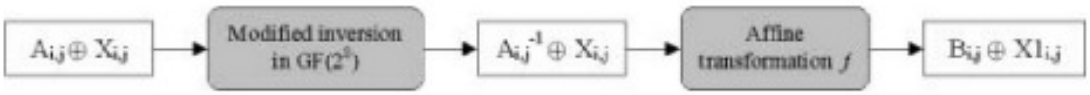

Figure 2.17: SubByte operation with masking countermeasure [12]

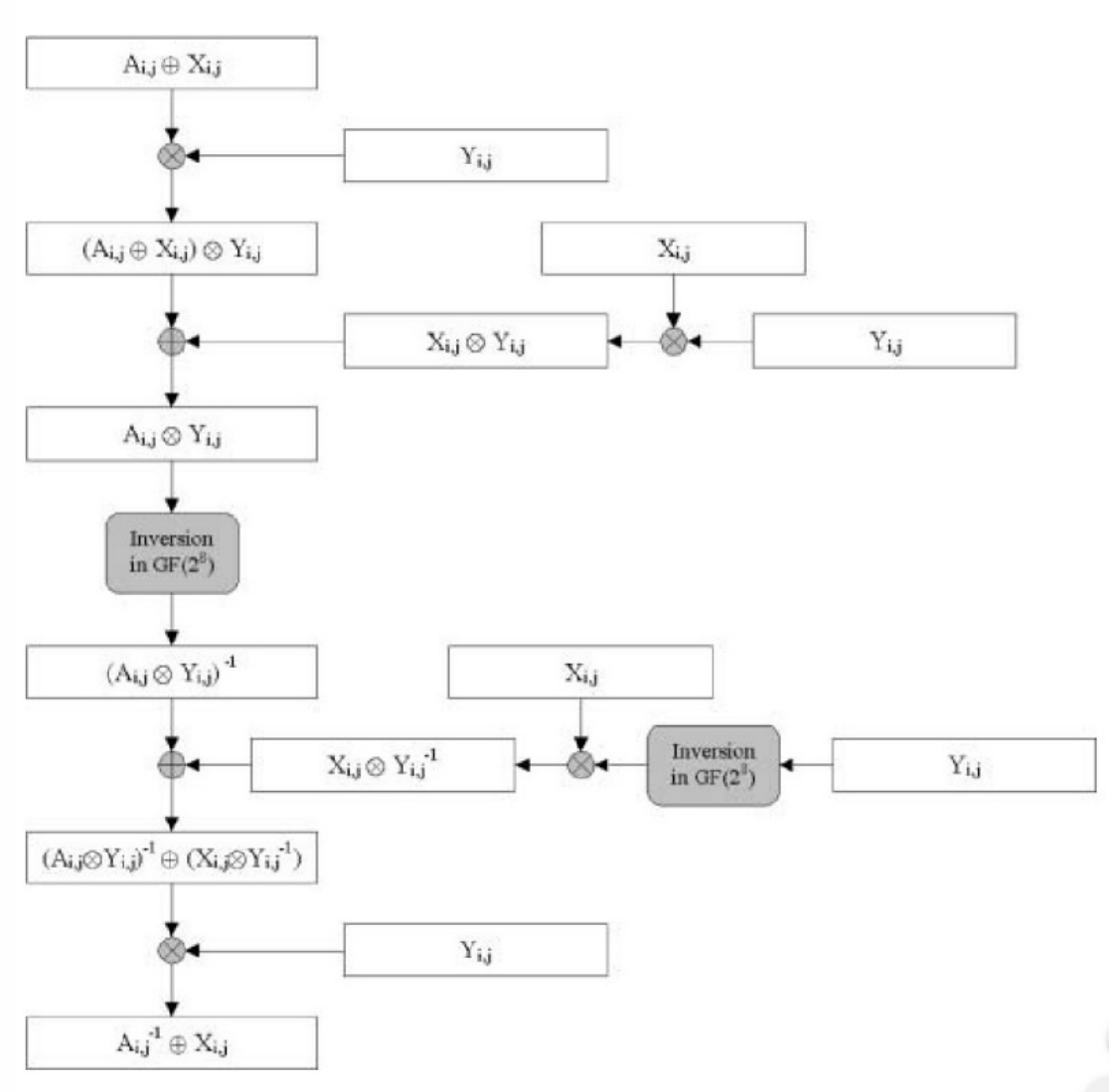

Figure 2.18: Detailed view of the masked inversion [12]

2.3.3.2 Masking against Higher Order DPA

The countermeasure discussed in the previous section is vulnerable to second order DPA attacks. The adversary may identify multiple samples in the trace set obtained that would allow him to remove the mask used in the encryption process by correlating a

combination function of the masked intermediate value and the mask applied.

Rivain et al proposed a generic d^{th}-order masking scheme, where d represents the number of masks applied to each key dependent intermediate value [14]. This results in $d+1$ shares that is used in the encryption algorithm. This countermeasure is based on the principle of secret sharing schemes. Each intermediate value is split into multiple shares each of which can be operated upon individually. The resultant shares can be recombined to retrieve the result of the operation on the secret. The splitting of the intermediate variable, x into d shares is illustrated below.

$$x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus ...x_d = x \tag{2.38}$$

In the above equation, d shares are randomly selected and x_0 is calculated to satisfy the equation. Splitting of the intermediate sensitive variable into shares ensures that any computation that takes place on the shares is independent of the sensitive variable.

The d^{th}-order masking scheme is vulnerable to an attack of $(d+1)^{th}$-order. This attack selects points which target the leakage related to the $d+1$ intermediate variables simultaneously. However, for $d \geq 3$, such attacks become impractical due to increased complexity and memory constraints. This makes a higher-order masking scheme a sound countermeasure.

A Brief Introduction to Artificial Neural Networks

3

Artificial Neural Networks (ANNs) are computing systems which possess the ability to recognize patterns, observe trends, analyze data and make predictions based on the data-sets. There is growing reliance on Neural networks in various disciplines such as speech [50], image and video processing [51], weather forecasting [52] and cyber-security [53]. ANNs have revolutionized many aspects of modern society and the fruitful interactions that are possible between Neural networks and research in various fields are proof to this claim. This chapter offers an insight into the working principles of a neural network. Section 3.1 provides a background of the building blocks of Neural Networks. Section 3.2 highlights different parameters that affect the performance of the neural network. Section 3.3 describe the hyperparameters that define the structure and training of neural networks.

3.1 Building blocks of Neural Networks

An Artificial Neural Network comprises of two building blocks, the neurons and the weights. Figure 3.1 illustrates a single neuron and n weights. An input is given on each weight and an external input in the form of bias is also applied to the neuron. The neuron computes a function on the bias, inputs and the weights and produces an output that is fed to an activation function. These functions are used to apply a transformation on the output of neuron so as to pass the information to the next neuron.

The function, f of the inputs x_i, weights w_i and the bias which is computed by the neuron is given below.

$$f(neuron) = \sum_{i=1}^{n} x_i w_i + bias \tag{3.1}$$

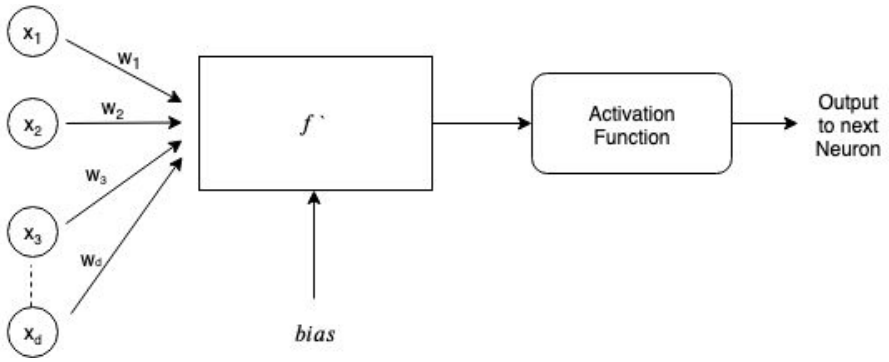

Figure 3.1: Structure of a simple ANN

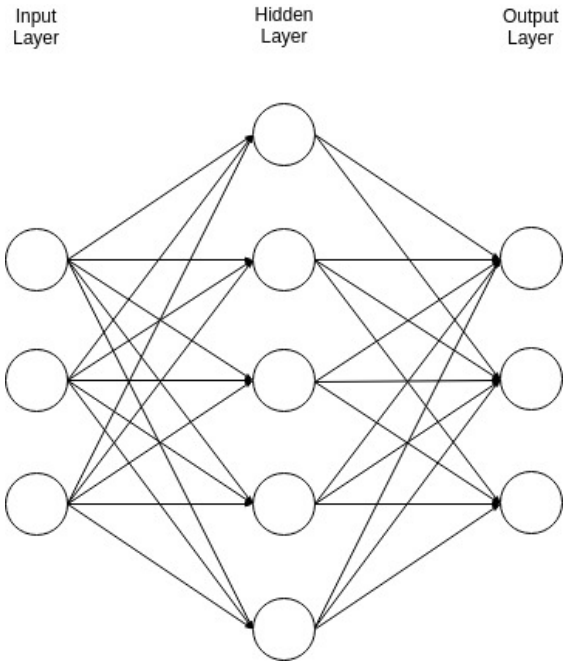

Figure 3.2: Structure of Multilayer Perceptron

3.2 Multilayer Perceptron

The neuron detailed in the previous section forms the basic building block of complex networks also known as Multilayer Perceptrons (MLP) [54]. An example of Multilayer Perceptron with a single hidden layer is illustrated in Figure 3.2. The MLP depicted consists of three layers namely the input layer, the hidden layer and the output layer. The input layer acts as an entry point for the inputs into the neural network. Except for the neurons in the input layer, every other neuron uses an activation function. The hidden layer is responsible for extracting information and features from the input data. The output layer presents the output decision or prediction of the neural network. MLPs are fully connected structures with each node in one layer connected to every node in the following layer.

MLPs are trained on a series of input-output pairs during which the weight and bias values of the network are modified in order to model the dependency between the input and the output. The error of prediction from each iteration of training is monitored and the parameters are changed accordingly in order to minimize the error. The back propagation algorithm is used to adjust the parameters relative to the error of prediction.

3.3 Hyperparameters

Hyperparameters are design parameters of the neural network which are initialized before training the model for a given set of inputs and outputs. These variables define the

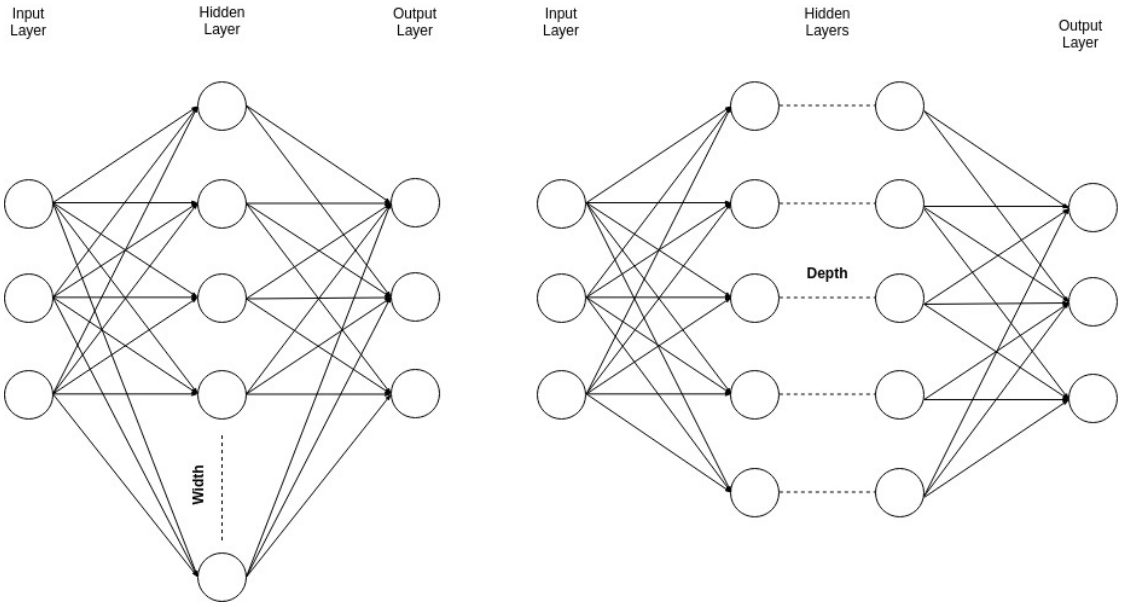

Figure 3.3: Width and Depth of a Multilayer Perceptron

structure (Eg: Number of neurons in the hidden layer) and influence the training algorithm of the neural network (Eg: The optimization and loss function used wile training the neural network). The hyperparameters can be classified into structural and training hyperparameters. The following sections explore different examples of the two classes.

3.3.1 Structural Hyperparameters

Structural hyperparameters include parameters that define the structure of the network. These parameters are width and depth of the neural network and the activation function used. They are explained in detail below along with illustrations.

3.3.1.1 Width of Neural Network

The width of the neural network refers to the number of neurons contained in each layer. Research in the field indicates that increasing width of neural networks shows a promising improvement in performance across networks of different depth [55]. It is also computationally more efficient to widen the network since multiplications required can be computed in parallel. In addition, the authors in [51] conclude that increase in width of a layer shows drastic improvement in performance of feature recognition systems.

3.3.1.2 Depth of a Neural Network

The depth of a neural network refers to the number of hidden layers contained in a neural network. Research indicates that deep neural networks provide a better generalization for a variety of tasks [56]. Increased depth enables networks to better capture variations in data. However, the computational demands of increased depth make it expensive to

implement in hardware and software. Figure 3.3 illustrates the width and depth of the MLP.

3.3.1.3 Activation Function

The activation function attached to the neurons in the MLP perform the task of transforming the weighted sum of the neurons of the previous layer to a suitable value for input to the next layer. Figure 3.4 illustrates commonly used activation functions [57]. Activation functions are used to introduce non-linearity in the neural network (with the exception of the identity activation function). This enables the network to detect and distinguish complex features. The activation function attached to a particular neuron allow the neuron to 'activate' when specific stimulus is given as input. In case of classification problems, the neuron corresponding to the correct output gets activated. Some activation functions are highlighted below.

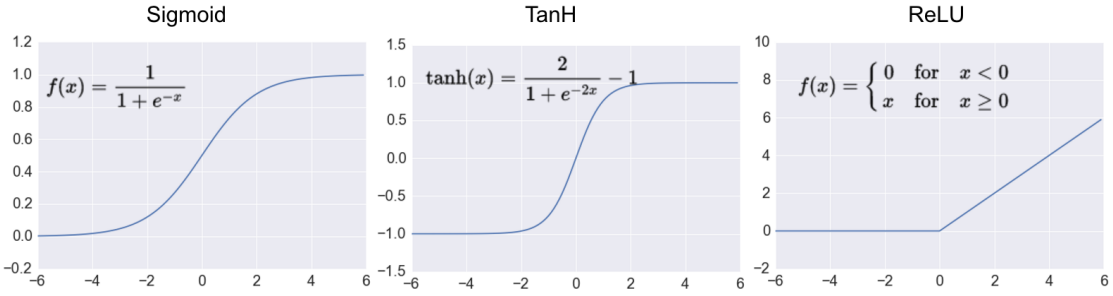

Figure 3.4: Commonly Used Activation Functions [57]

- **Identity**. The identity unit is a linear activation function that takes input arguments and returns them unchanged. In mathematical terms, the activation function, f for an input, x can be described to be of the form $f(x) = x$.

- **ReLU**. The Rectified Linear Unit activation function takes input arguments and returns the argument unchanged if it is a positive value. The function returns a zero for negative input values. In mathematical terms, the activation function, f for an input, x can be described to be of the form $f(x) = max\{0, x\}$.

- **Hyperbolic tangent**. The Hyperbolic tangent activation function takes input arguments and applies the hyperbolic tangent function on them in the range of $(-1, 1)$. In mathematical terms, the activation function, f for input, x can be described to be of the form $f(x) = tanh(c) = \frac{(e^x - e^x)}{(e^x + e^x)}$.

- **Sigmoid**.The Sigmoid tangent activation function takes input arguments and applies the hyperbolic tangent function on them in the range of $(0, 1)$. In mathematical terms, the activation function, f for input, x can be described to be of the form $f(x) = \frac{1}{(1+e^{-x})}$.

3.3.2 Training Hyperparameters

Training hyperparameters include functions involved in the training of the neural network. The parameters are the initialization, regularization and loss functions. Each function is explained in detail below along with equations.

3.3.2.1 Initialization

The initialization function involves setting the values of the weights and bias of the neural network before training. Research indicates that some initialization functions enable faster training of the neural network model. Following are some basic initialization techniques used to initialize the parameters of the neural network.

- **Zero initialization**. This initialization function sets all the weights and biases to zero. This makes every neuron in the hidden layer symmetric throughout the training process. This results in all neurons having the same value of weights after iterations of training. This results in the complexity of the neural network to be equivalent to that of a single neuron. Thus, zero initialization is not suitable for classification problems.
- **Random initialization**. This initialization function sets random values to the weights and biases of the neural network. Care must be taken to set the range of random value set to the weights. High values of weights can result in slow training of the networks while weights initialized with extremely low values leads to results similar to zero initialization [58].
- **He initialization**. This initialization function is an extension of the random initialization function. In this method, the weights of a given layer are initialized taking into consideration the width of the previous layer. This is defined in Equation 3.2. This factor is multiplied with the randomness function and results in faster training of the network [59].

$$\text{He Initialization Factor} = \sqrt{\frac{2}{\text{Size of Previous Layer}}} \tag{3.2}$$

- **Xavier initialization**. This initialization function is a modification to the He initialization function. The initialization factor as shown in Equation 3.3 [60].

$$\text{Xavier Initialization Factor} = \sqrt{\frac{1}{\text{Size of Previous Layer}}} \tag{3.3}$$

3.3.2.2 Loss Function

The loss function is tasked with calculation of the error between the predicted output value produced by a network and the actual value provided by the user from the input data-set. The neural network aims to minimize the loss function in the training phase in order to improve prediction accuracy. Some commonly used loss functions are highlighted below. In the following equations, it is assumed that N represents the number of samples, y_i represents the actual value present in the training data-set and a_i represents the predicted value.

- **Quadratic Error**. The quadratic error loss function is implemented by calculating the sum of the absolute difference between the predicted and actual output for the entire data-set. This function is defined in Equation 3.4.

$$\text{Quadratic Error} = \sum_{i=1}^{N} |\mathrm{a}_i - \mathrm{y}_i|^2 \tag{3.4}$$

- **Cross Entropy**. The cross entropy loss function is implemented by calculating the difference between probability distributions of the predicted and the actual output of the data-set. It is commonly used in classification problems whose output is the probability value. The loss function decreases when the predicted probability distribution approaches the actual probability distribution. It is also referred to as the log loss function. The loss function applies heavy penalization to the network for confident and incorrect predictions [61]. This function is defined in Equation 3.5.

$$\text{Cross Entropy Error} = \sum_{i=1}^{N} |\mathrm{a}_i log(\mathrm{y}_i) + (1 - \mathrm{a}_i) log(1 - \mathrm{y}_i)| \tag{3.5}$$

3.3.2.3 Regularization

The regularization function is tasked with preventing the neural network from overfitting to the training data. Overfitting is a modeling error that causes the trained neural network to map too closely to the training data during the training stage and not generalize enough to possess the ability to predict correctly in the inference stage. Regularization functions are common methods to prevent this phenomenon and improve the performance of the neural network. Some commonly used regularization techniques are highlighted below.

- **L1 / L2 Regularization**. The regularization function ensures an upper limit on the weights of the neurons in the network. Large weights leads to poor performance and overfitting of the network. The L1/L2 regularization is implemented by adding a term to the cost function as illustrated in Equation 3.6. This ensures that the neural network trains with smaller weights preventing overfitting of the trained model [62]. The regularization parameter, λ is a hyperparameter that is used for tuning the degree of regularization. In case of L1 regularization, L is substituted with the absolute sum of all the weights. Here, higher value of weights are penalized during the next iteration of training. Similarly, in L2 regularization, L is substituted with the square of the absolute sum of all the weights. This is also known an weight decay since the weights are forced to converge to a minimum.

$$\text{Cost Function} = \text{Loss Function} + (\lambda \times L) \tag{3.6}$$

- **Early Stopping** The early stopping function avoids overfitting by setting a threshold on the performance of the neural network during training. This is implemented by testing the data on a validation dataset at the end of each iteration. The training is stopped on achieving the threshold performance desired on the validation data.

Early stopping is considered to be a type of "implicit" regularization, wherein no modifications are made to the loss function. This regularization technique is more suitable on small networks.The research done in [63] highlight that early stopping is ineffective on deep convolutional neural networks that perform the task of object classification.

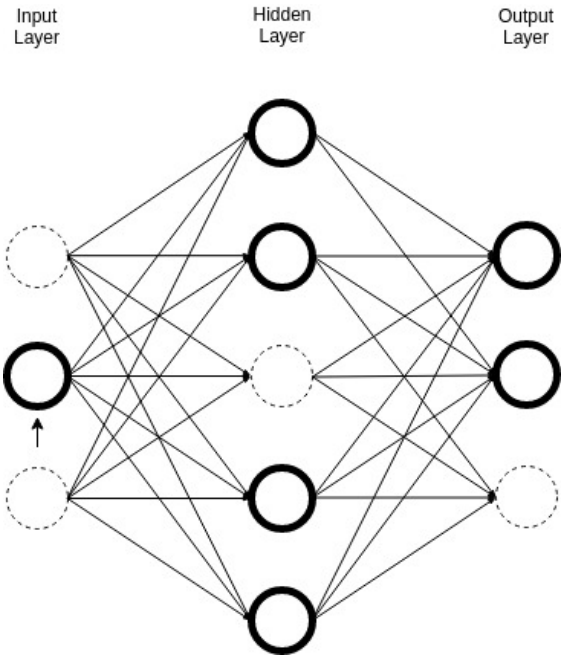

Figure 3.5: Dropout as a Regularization Technique

- **Dropout** The dropout function avoids overfitting by dropping randomly selected neurons and their associated weights from each layer during each iteration of the training phase. Figure 3.5 illustrates a neural network using dropout as a regularization technique. The shaded nodes indicates neurons which are dropped and the darkened neurons are used in the current iteration of the training phase. The function is parameterized by D which defines the number of dropped neurons in each layer and probabiltity p which defines the probability that a particular node is dropped out of the network. This function ensures that neurons are not codependent on each other and the individual features of every neuron are used to generalize to the data [64]. This forces each neuron to learn features with a random subset of other neurons in the network. This however leads to more number of iterations required to train the network.

3.4 Backpropagation

The backpropagation algorithm is tasked with iterative updation of the weights, w_i and biases, *bias* of the training network based on the output of the selected loss function, E. This is implemented by computation of the partial derivative of the output of the loss

function with respect to the weights of the neural network. In this manner, the algorithm computes the contribution of each parameter to the error. This is represented by $\frac{\partial E}{\partial W_{i,j}}$ and $\frac{\partial E}{\partial bias_i}$ for neuron i. The optimization function is parameterized by the learning rate defined by β. The steps involved in the backpropagation algorithm is highlighted in detail below.

1. The error, E is obtained from the result of the loss function.

2. The gradient of the error obtained is calculated with respect to the values of the bias and weights of the neural network. This is indicated in Equation 3.7.

$$\text{Gradient Error} = \frac{\partial E}{\partial W_{i,j}} \tag{3.7}$$

3. The weights $W_{i,j}^{old}$ in each layer are updated to a value $W_{i,j}^{new}$ by an amount that is proportional to the negative gradient of the error obtained. This is indicated in Equation 3.8. The rate of updating is defined by the learning rate, β. The first term of the gradient defines the derivative of the total error with repsect to the selected activation function, A. The following term of the gradient defines the derivative of the activation function with respect to the output of the corresponding neuron y_i. The final term defines the derivative of the output with respect to the corresponding weight associated with the neuron. These terms are combined in order to calculate the gradient for each weight. This is indicated in Equation 3.9.

$$W_{i,j}^{new} = W_{i,j}^{old} + \beta \times \frac{\partial E}{\partial W_{i,j}^{old}} \tag{3.8}$$

$$\frac{\partial E}{\partial W_{i,j}} = \frac{\partial E}{\partial A} \times \frac{\partial A}{\partial y} \times \frac{\partial y}{\partial W_{i,j}} \tag{3.9}$$

4. This previous step is continued for the weight set in every layer through back propagation of the error from the final year to the current layer. In this manner, the weights are adjusted corresponding to the loss every neuron is responsible for.

5. It is important to note that the loss of the neurons contained in the final layer is equivalent to the accumulated loss of the entire network. Also, the neurons in the input layer are not involved in the backpropagation process since they have no weight sets associated with any preceding layer.

3.5 Training and Inference of Neural Networks

3.5.1 Training and Test Data

The input data-set is divided into training and testing data. In the case of supervised learning, the training data-set includes both the input data, as well as the corresponding output (label). The testing data includes only input data. The training data is typically

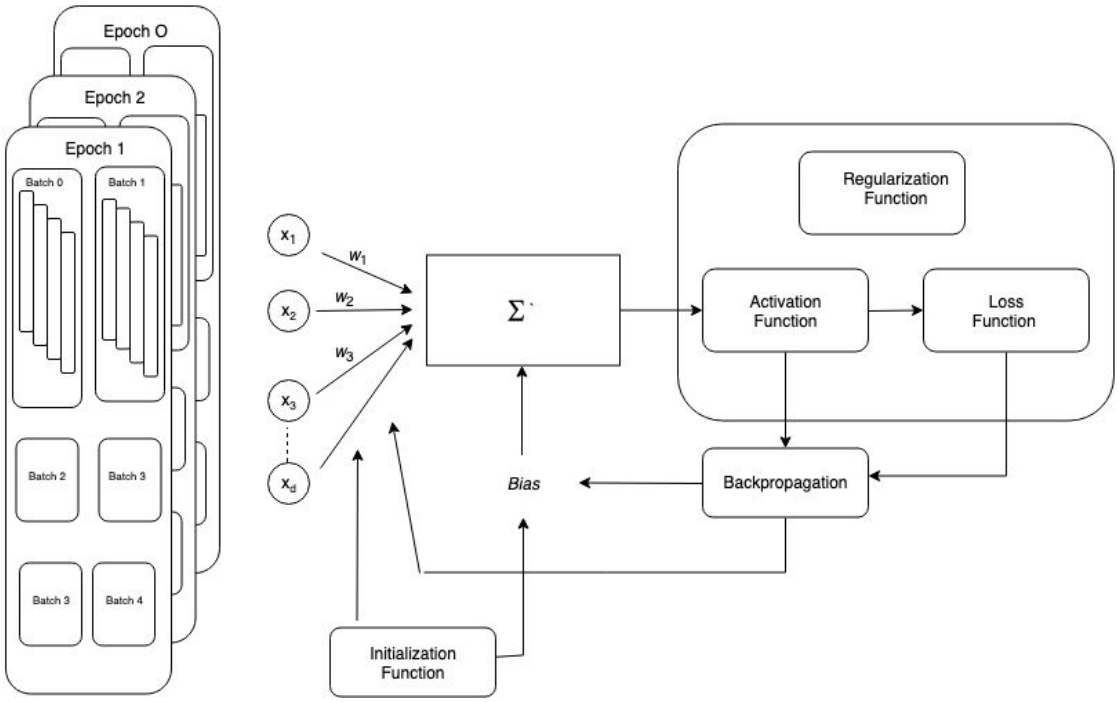

Figure 3.6: Training of Neural Networks

larger in size than the test data. This method of learning is called supervised learning, since the label for the training data-set is used as a learning guide for the neural network during the training phase. The hypothesis is used to predict unknown labels for the test data in the inference stage. The performance of the neural network can be measured by analyzing its prediction accuracy for test data in the inference stage.

3.5.2 Training of Neural Networks

Figure 3.6 provides an illustration of the steps involved in the training of a neural network. The training data is further divided into smaller groups called batches and are applied to the neural network iteratively for a number of epochs. The weights and biases of the neural network are updated in order to map the function represented by the training data. These values are updated after a batch of training data has been applied to the neural network. Thus, each iteration of training involves multiple batches applied to the neural network and updating the weights and bias at the end of each complete batch. The training procedure involves multiple such iterations known as epochs. The number of epochs is represented by O. The steps involved in the training process are highlighted below.

1. The training and structural hyperparameters of the neural network are selected prior to the training process.

2. The weights are bias of the neural network are initialized using the selected **initialization function**.

3. Batches are created from the input training data set and assigned to each epoch of the training process.

4. The first batch is applied to the neural network.

5. The error of the neural network for the batch is computed using the selected **loss function**.

6. The weights and bias of the neural network is adjusted depending on the error propagated through the network and the **back propagation function**.

7. The selected **regularization function** for the epoch is applied during the training process of each epoch.

8. The steps 3-5 are repeated for all batches.

9. The steps 3-7 are repeated for all epochs.

3.5.3 Inference of Neural Networks

The inference procedure involves testing the trained neural network against new data and measuring the obtained performance. The weight sets of the trained neural network are frozen and no longer updated in this stage. This procedure also involves 'pruning' the neural network by removing the neurons which are not activated during the training process. These neurons are indicated by zero-valued incoming and outgoing weights to the network. In extreme cases, multiple layers of the neural network can be fused to form a single layer [65]. It is observed that in such cases, negligible change in accuracy of prediction is observed.

Neural Network Based Countermeasure

4

This chapter explains the key concepts behind the implementation of the Neural Network as a countermeasure (S-Net) . First, Section 4.1 highlights the motivation behind using neural networks as a countermeasure against power analysis attacks. Section 4.2 provides an in-depth design methodology of the S-Net. Section 4.3 describes optimizations done on the baseline design to create an optimal design of the S-Net.

4.1 Motivation of Design

Power Analysis techniques relies on exploiting the correlation between the observed power consumption of CMOS circuits and the data being stored and operated on by the circuit. An adversary makes use of power models to convert an hypothesis on an intermediate value to a hypothesis on the power consumption. Two commonly used leakage models is given below.

- Hamming Weight (HW) Model. Representation of the number of bits set to one in a given intermediate value. The variation of power consumption for Hamming Weights is shown in Figure 4.1. An example is illustrated below.

Input Intermediate Value to *SubByte* (Hex)	$= 0 \times 86$
Output Intermediate Value from *SubByte* (Hex)	$= 0 \times 44$
Output Intermediate Value from *SubByte* (Binary)	= 0110 0110
HW(Output Intermediate Value from *SubByte*)	= 4

- Hamming Distance (HD) Model. Representation of the number of transitions or bitflips that takes place during an operation on a given intermediate value. The variation of power consumption for Hamming Distances is shown in Figure 4.2. An example is illustrated below.

Input Intermediate Value to *SubByte* (Hex)	$= 0 \times 86$
Output Intermediate Value from *SubByte* (Hex)	$= 0 \times 44$
Input Intermediate Value from *SubByte* (Binary)	= 1000 0110
Output Intermediate Value from *SubByte* (Binary)	= 0110 0110
HD(Intermediate Value from *SubByte*)	= 3

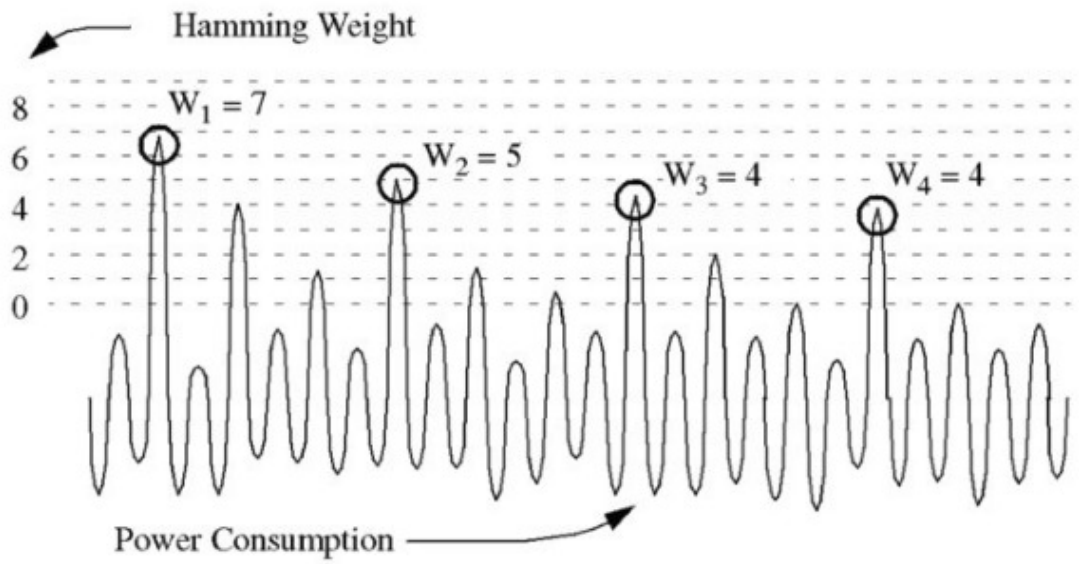

Figure 4.1: Variation of Power Consumption for Different Hamming Weights

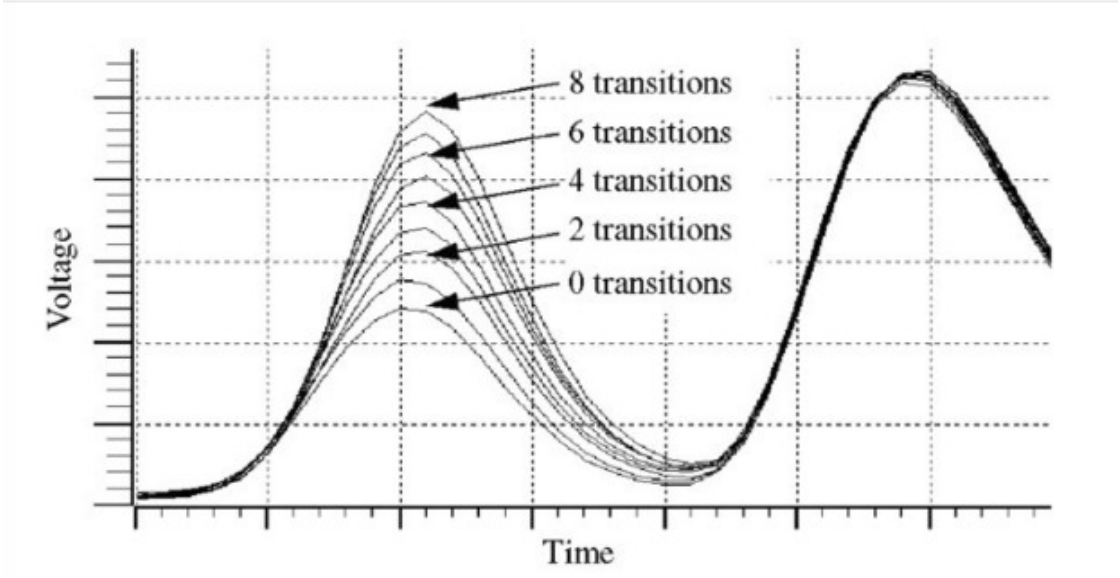

Figure 4.2: Variation of Power Consumption for Different Hamming Distances

Leakage models are used to quantize the power consumption for a given operation executed on the system. They allow the adversary to predict behaviour of power consumption with minimal effort. Figure 4.1 and Figure 4.2 illustrates the quantization of simulated power consumption signals using the Hamming Weight and Hamming Distance leakage model respectively [66].

This quantization of power consumption with the leakage models also assumes the

following relationships. The estimated power consumption while processing the intermediate variable is indicated by PC.

$$
\begin{aligned}
&\text{Intermediate Value}_1\text{(Hex)} && = 0 \times 02 \\
&\text{Intermediate Value}_2\text{ (Hex)} && = 0 \times 03 \\
&\text{Intermediate Value}_3\text{ (Hex)} && = 0 \times 0F \\
&\text{Intermediate Value}_1\text{(Binary)} && = 0000\ 0010 \\
&\text{Intermediate Value}_2\text{ (Binary)} && = 0000\ 0011 \\
&\text{Intermediate Value}_3\text{ (Binary)} && = 0000\ 1111 \\
&\text{HW(Intermediate Value}_1\text{)} && < \text{ HW(Intermediate Value}_2\text{)} \\
&\text{PC(Intermediate Value}_1\text{)} && < \text{ PC(Intermediate Value}_2\text{)} \\
&\text{HW(Intermediate Value}_3\text{)} && = 2 \times \text{ HW(Intermediate Value}_2\text{)} \\
&\text{PC(Intermediate Value}_3\text{)} && \approx 2 \times \text{ PC(Intermediate Value}_2\text{)}
\end{aligned}
$$

The proposed design seeks to break the linear dependency of power analysis techniques on the commonly used leakage models with the help of a neural network. The following equations indicate the result of application of the leakage model on the input and output to the designed neural network.

$$
\begin{aligned}
&\text{Input Intermediate Value to } \mathit{SubByte} \text{ (Hex)} = 0 \times 86 \\
&\text{Output Intermediate Values (Integer Array)} = -7948, 4510, -18468, -5480, \\
&\qquad\qquad -11789, 8202, -6721, -3299 \\
&\text{Output Intermediate Value (After Conversion)} = 0, 1, 0, 0, 0, 1, 0, 0 \\
&\text{Output Intermediate Value from } \mathit{SubByte} \text{ (Hex)} = 0 \times 44
\end{aligned}
$$

The designed neural network outputs an integer corresponding to each output bit. The low bit is indicated by a negative output integer and a high bit is indicated by a positive output integer. Each combination of eight bit outputs have an unique integer representation which do not comply with the previously stated relationships. Thus it can be hypothesized that the leakage models cannot be used to model the power consumption and the implementation must be secure against side-channel attacks. This thesis aims at proving this hypothesis.

4.2 Design Methodology

Figure 4.3 illustrates the high-level view of the programming environment used to train the neural network and obtain the weight sets required to perform the mapping. Each component of the platform is explained in detail below.

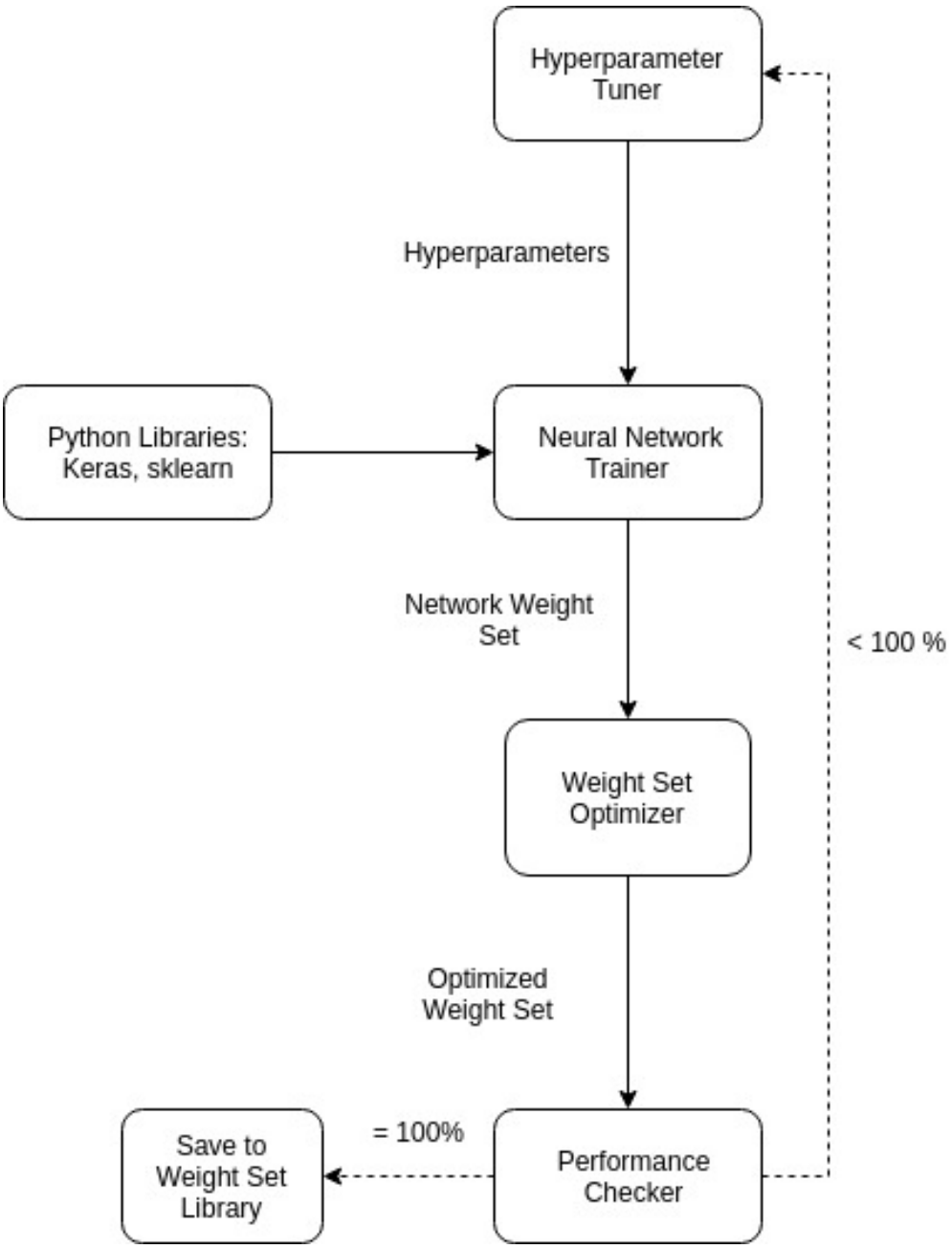

Figure 4.3: Programming Environment

- **Python Libraries: Keras, sklearn**. Open source libraries with functions for building and evaluating neural network models with different structural and training hyperparameters. The sklearn library provides pre-processing, training and inference functions required to train the network for an input-output pair [67]. The Keras library is used to export weight sets and network architectures which may be used on other platforms. The structure of the network to be trained and

tested can be stored in a *.json* format using Keras [68].

- **Hyperparameter Tuner**. The hyperparameter tuner changes the value of the structural and training hyperparameters of the target architecture according in a predefined manner. This is primarily used to perform a design space exploration (DSE) on the hyperparameters of the neural network. The results of this DSE is then evaluated to implement an optimal structure capable of performing the entire mapping.

- **Neural Network Trainer**. The trainer makes use of the library functions and the hyperparameters returned by the hyperparameter tuner in order to train the neural network with the 256 pairs of input and output. This is done for a fixed set of iterations and returns a weight set to the weight set optimizer in the form of *numpy* arrays.

- **Weight Set Optimizer**. The weight set optimzer attempts to apply multiple optimization functions to the weights of the neural network. The optimzations include limiting the weights to a fixed range, conversion of weight sets from floating point to fixed point, dropping weights and neurons which do not contribute significantly to the output, etc. This is highlighted in detail in Section 4.4.

- **Performance Checker**. The performance of the neural network is checked in this module. It is required that the neural network performs the correct mapping for all possible inputs. If not, the hyperparameters are tuned further until a full mapping is achieved.

- **Weight Set Library**. This module parses the weight set from the architecture returned by the Python library and saves the information in the weight set library to be used as a countermeasure.

4.3 S-Net Design

The Look Up Table used in the *SubByte* operation is replaced with a neural network that performs the same mapping. The neural network designed is required to replicate the *SubByte* operation for all possible inputs. Thus, it it is required that neural network overfit over all possible inputs and outputs so as to perform the required mapping with full accuracy.

The S-Net is designed with eight inputs and outputs. Each neuron in the input and output layer are activated corresponding to the representation of the numbers in binary. For example, an input of $A8$ would turn on the first, third and fifth neuron corresponding to 1010 1000. Given this input, the neural network would activate the first, third, fourth and fifth neuron in the output layer, corresponding to $5C$, which is the result of the *SubByte* operation. The activation is triggered if the output integers produced in the final layer is positive. This is illustrated in Figure 4.4.

The sign of the values of the neurons in the output layer was used to set the output bits to 1 and 0. A positive integer indicated a 1 and a negative integer indicated a 0. It

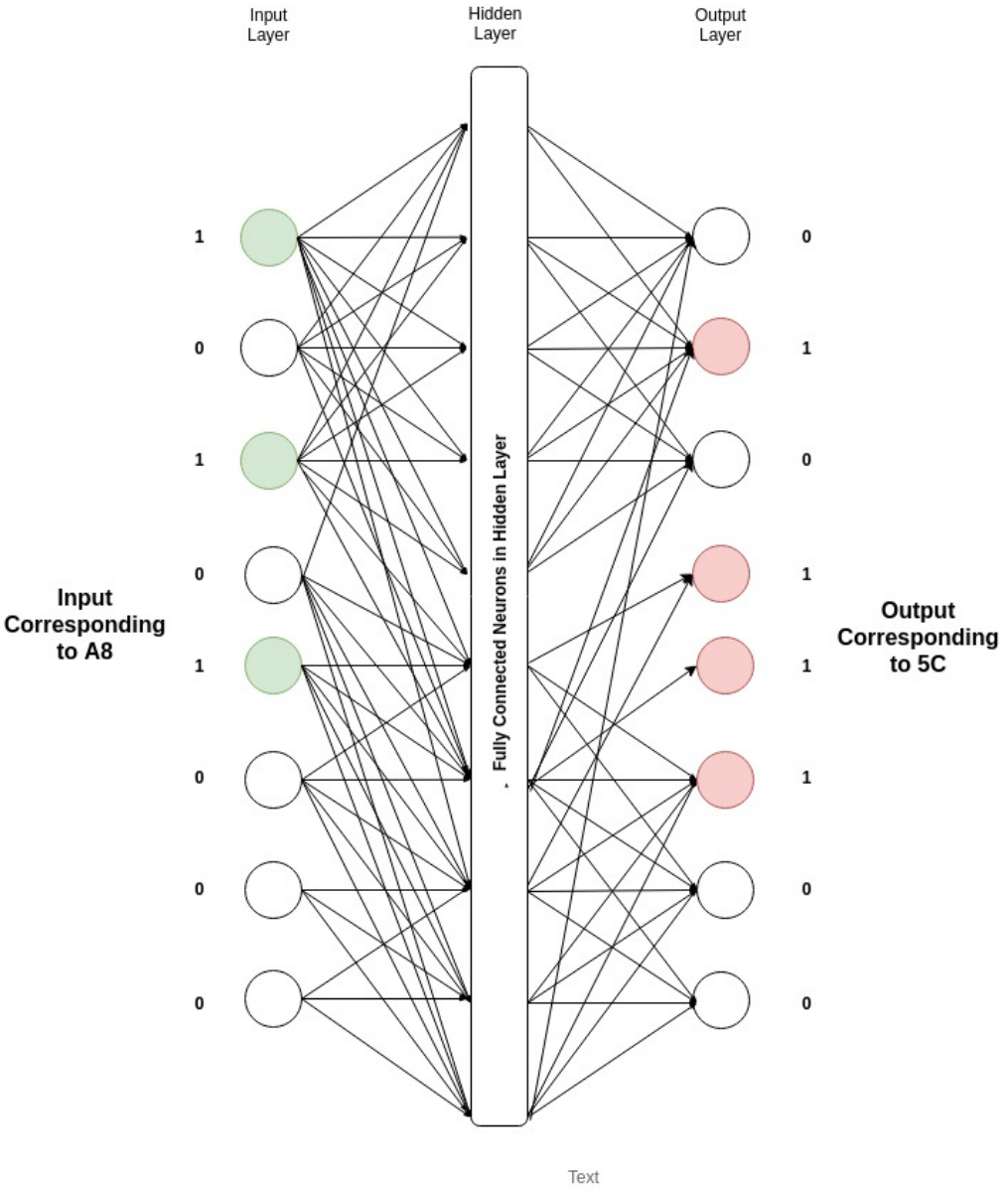

Figure 4.4: S-Net Design

was observed that the values of the integers had a wide range and this representation of numbers as integers makes it possible to break the leakage models that power analysis techniques rely on.

4.4 Optimization of Design

Optimization functions were implemented on the weight sets in order to ensure an implementation which is computation and memory efficient. This function was applied after

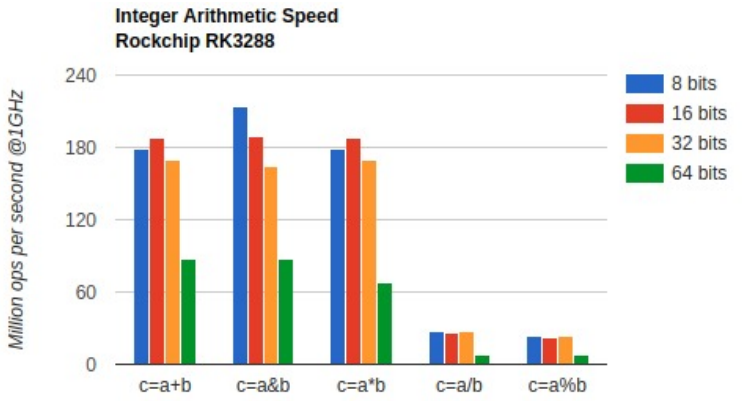

Figure 4.5: Integer Arithmetic performance

training the neural network for all 256 input and output pairs and ensuring complete mapping of the *SubByte* function. The optimization techniques are highlighted in detail below.

- **Data Representation**. The result of the training phase of the neural networks produces a floating point weight set. Figure 5.2 indicates the performance of calculations on floating point arithmetic versus integer arithmetic on am ARM Cortex based target board [69]. It is clear that integer operations show a significant performance gain on comparison to floating point numbers. The advantage of implementing the neural network using fixed point arithmetic is apparent. For this reason, the weights of the neural network are typecasted to the integer data type after the training phase. It is then checked if the integer data type does not affect the performance of the mapping. On achieving complete mapping, the integer weight set is then saved to the weight library.

- **Range of Weights**. The result of the training phase of the neural networks produced a distribution of weights over a wide range. Implementation of a neural network in hardware and software would be more optimal if the weight set can be represented by distribution of weights bounded by an upper limit of bits. For this reason, the weights of the neural network are forced to be in the range of numbers which can be represented by 16 bits allowing faster operations and lower memory overhead.

- **Removal of bias**. The result of the training phase of the neural networks produced a set of weight and bias units for each layer. It was observed that the neural network was capable of performing mapping with the same performance without the aid of bias units in the output layer. Thus, the bias units of the output layer was set to zero. It was also observed that bias units in the hidden layers below a particular threshold did not contribute significantly to the activation of the neuron in that layer. Thus, bias units obtained with a value below the threshold were set to zero.

- **Multiplication Threshold**. The result of the training phase of the neural net-

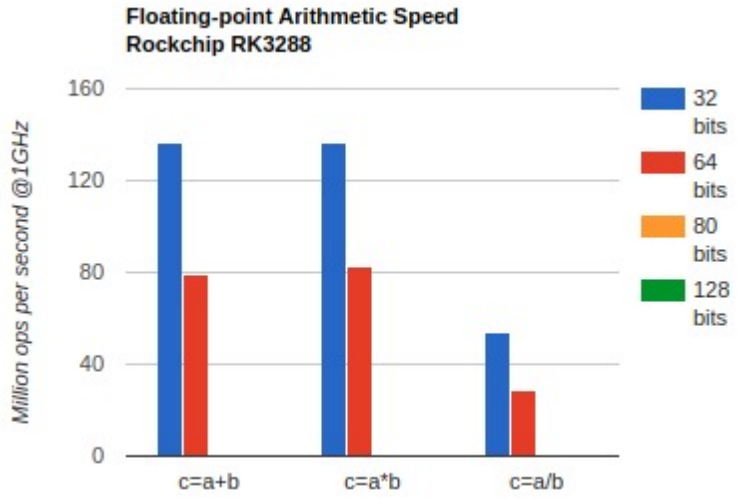

Figure 4.6: Floating point Arithmetic performance

works produced neurons and associated weights which did not contribute significantly to the activation of the output in the output layer. For this reason, in the inference phase, a threshold was set on the products produced in the hidden layer and the weights. The values below the threshold does not participate in the computation of the final layer of outputs. This results in lesser computational overhead of the architecture.

Validation and Results Analysis

5

This chapter highlights the experimental platforms used to acquire traces and perform side-channel analysis on the target. It also explores the implementation details of the S-Net on the target boards. The experimental platform used to validate the effectiveness of the S-Net as a countermeasure is highlighted in Section 5.1. Section 5.2 highlights the implementation details of the S-Net on the targets. Section 5.3 provides the evaluation of the security of the implementation against various attacks. Lastly, Section 5.4 provides a perfomance analysis of the countermeasure in comparison to the unprotected implementation of AES.

5.1 Experimental Platform

Figure 5.1 illustrates the high-level view of the experimental platform used to acquire power traces, evaluate and validate the security of the software implementation of the

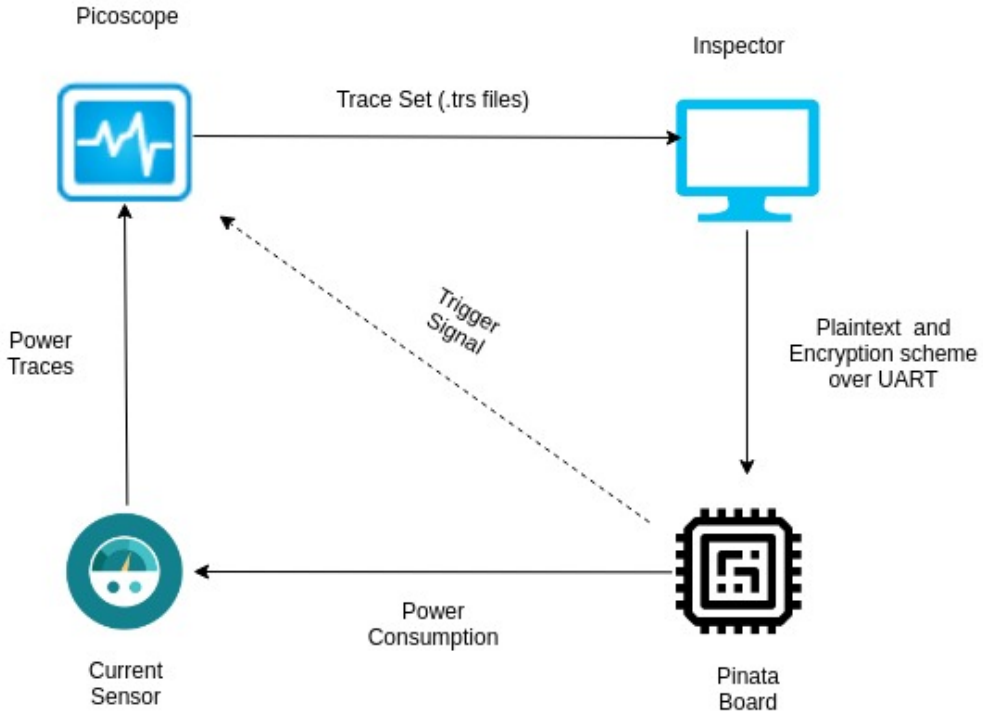

Figure 5.1: High level overview of the experimental set up for software implementation of AES

Oscilloscope	Picoscope 3206D
Input Channels	2 channels
Bandwidth	200MHz
Input Ranges	+/- 20 mV to 20 V
Maximum Sampling Frequency	1 GS/s
Maximum Capture Rate	100,000 waveforms/second
Source	Analog and Digital Channels channels, EXT trigger
Trigger Modes	None, auto, repeat, single, rapid

Table 5.1: Technical specifications of Picoscope 3206D

Figure 5.2: Picoscope 3206D [70]

S-Net. Each component is highlighted in detail in the following subsections.

- **Picoscope 3206D.** Power traces for this experiment are acquired from the target board using the Picoscope. Each power trace is stored along with the corresponding plaintext being encrypted. The beginning and end of each encryption is identified using a trigger signal produced by the target board. The Picoscope is connected and powered via USB to a host system which allows real time acquisition and analysis of traces. The traces is stored in a *.trs* format. The Picoscope software also presents the user with multiple options such as int / float / double representation of power values while storing the power trace. Some specifications of the oscilloscope are highlighted in Table 5.1.

  The Picoscope firmware directly interfaces with the Inspector software used to analyze the power traces. This allows ease of acquisition and analysis taking into consideration that a large number of power traces need to be acquired. The sampling frequency of acquisition is dependent on the target under consideration. Since the Pinata board runs at a clock speed of 168Mhz, a sampling frequency of 1GS/s is used.

- **Pinata Target board** The Pinata board is the target microcontroller for the software implementation of the S-Net. It is a development board manufactured by Riscure for the purpose of implementing side-channel attacks and countermeasures.

Target Board	Pinata
Processor	ARM Cortex-M4F
Maximum Clock Speed	168MHz
SRAM memory	196 Kbytes
Crypto Engine	TRNG, cryptography engine for encryption algorithms
Power	3.3V input

Table 5.2: Technical Specifications of the Pinata Board

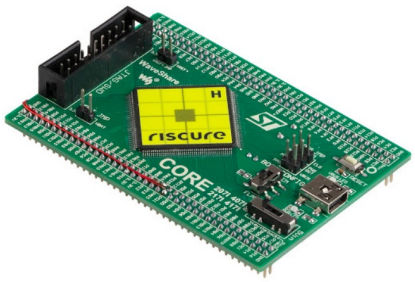

Figure 5.3: Riscure Pinata Board [71]

The source code and IDE required to program the board with custom implementations are provided along with the board. The current is measured using a current sensor as highlighted in the next section. The Pinata board also contains a trigger signal output from a GPIO pin which triggers at the beginning and end of every encryption. This makes the board an ideal, low cost solution to implement passive side-channel attacks. Some specification of the Pinata board are highlighted in Table 5.2. The plaintext to be encrypted along with the encryption scheme to be used is transmitted from the host system using the UART interface. The current sensor is connected in series to the 3.3V input power supply. Any modifications to the firmware on the board is downloaded on to the board via the USB interface. The board is depicted in Figure 5.3.

- **Current probe** The current probe provides low-noise, high quality measurement of the instantaneous power consumption of the target board. The current probe used is depicted in Figure 5.4. It is a passive device capable of measuring electric currents, which are directly proportional to the power consumed by the target board. The current probe is inserted in the power supply line of the target board and is used in combination with an amplifier to measure and amplify current variations in the order of 1000 MHz. Some technical specifications of the current sensor are shown in Table 5.3. Power lines of embedded systems usually produce noisy signals with a very wide bandwidth. The current probe is effective with devices that involve a cryptographic processor that is shielded or involves cooling elements. Figure 5.5 shows an overview of the setup and connection of the current probe.

Current Probe	Built-in Tektronix CT1 current probe
Maximum Bandwidth	1 GHz
Frequency range	0.1 to 2500MHz
DC Power	12 V
Gain	25 dB at 1000MHz

Table 5.3: Technical Specifications of the Pinata Board

Figure 5.4: Riscure Current Probe [72]

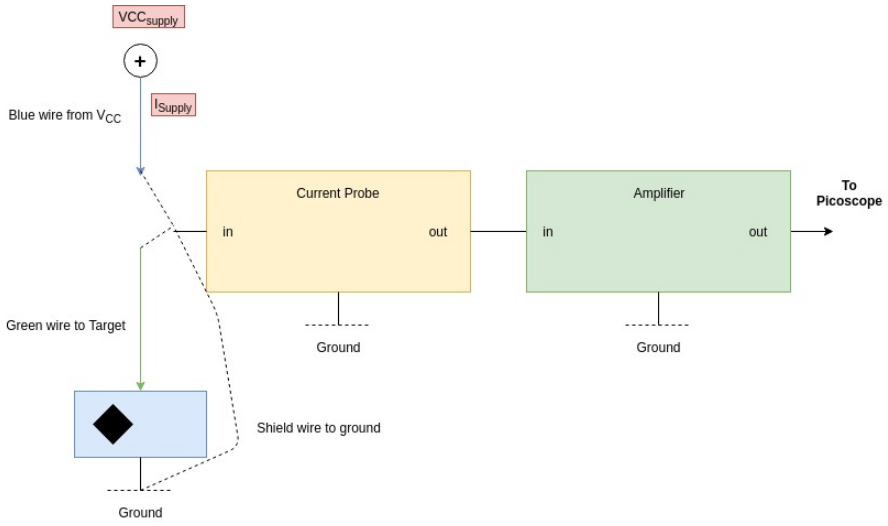

Figure 5.5: Setup and connection of the current probe

- **Inspector**. The Inspector is a side-channel analysis program developed by Riscure B.V [73]. It provides a library of functions that allow an user to interface with the data acquisition models such as the Picoscope and the Pinata board. This strong hardware integration allows easy configuration of both the Picoscope and the Pinata board from a single GUI-based platform, tremendously reducing time taken to perform security evaluations. It supports commonly used algorithms such as AES, DES, 3DES, RSA, RC4 and also provides tools to perform side-channel

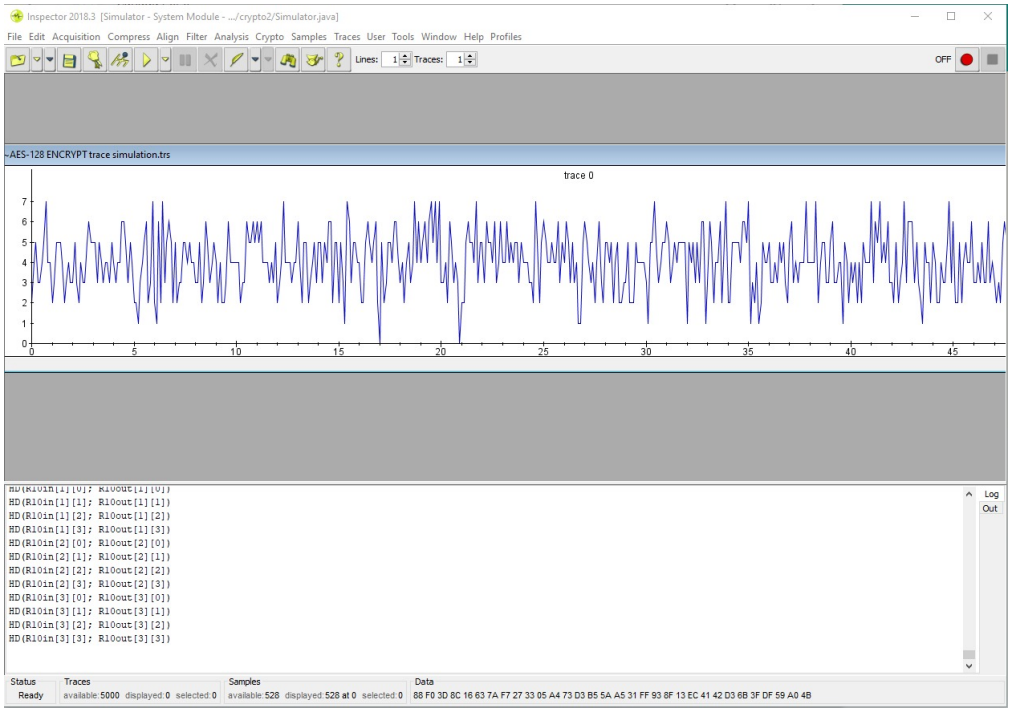

Figure 5.6: Inspector Interface

and statistical analysis on implementations running these algorithms. It is capable of performing modifications to the trace sets such as realignment, windowed resampling, filtering, etc. For this experimental setup, the Inspector software is used to run Differential and Correlation Power analysis on the protected and unprotected software implementation of AES executed on the Pinata. Figure 5.6 shows a display of the Inspector interface.

- **Trace Sets**. Riscure uses the *.trs* file format to store and read the acquired traces from the hard drive. An open source Python library is made available [74] for users to perform the own side-channel and statistical analysis on these tracesets. Every trace acquired by the Picoscope has an associated header which contains the plaintext to be encrypted, the number of samples, the trace offset etc. An illustration of the trace set visualized in the Inspector software is provided in Figure 5.7.

5.2 Implementation Details

The S-Net was implemented on software to be executed on the Pinata evaluation board. The architecture of the network including the appropriate hyperparameters was selected from the results of the training phase in Python. These details are highlighted in the following subsections.

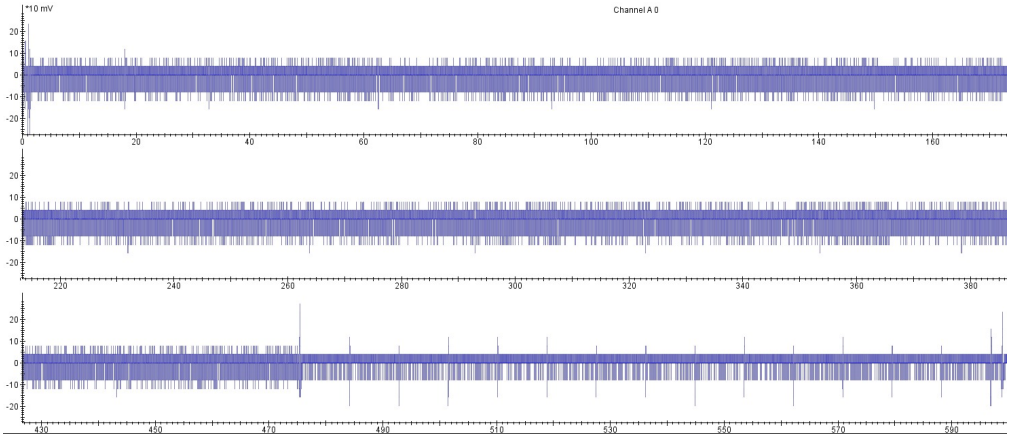

Figure 5.7: Visualization of trace sets of the S-Net during encryption

5.2.1 Width and Depth of the S-Net

The neural network was trained iteratively for different width and depth parameters. It was observed that a minimum number of neurons and hidden layers were required to achieve complete mapping of the *SubByte* function. Table 5.4 illustrates the width and depth parameters that were able to successfully map the function. The network is represented by $a : b : c : d : e$ where each value indicates the number of neurons in each layer. The first and last number indicated the number of neurons in the input and output layer respectively and a zero value indicates the absence of the hidden layer. The first set of width and depth parameters were selected since increased depth with a smaller width parameter is more computationally expensive and memory expensive. These calculations are indicated by the number of arithmetic operations in the table.

5.2.2 Activation Function

The neural network was trained with different activation functions. Literature indicates that the Sigmoid function and Hyperbolic Tagent activation functions are capable of mapping non linear relationships with ease. However, it was also observed that the Rectified Linear Unit (ReLU) function was successfully in performing the same mapping with the selected architecture. It is also more computationally efficient in using the ReLU function since it ensures that only neurons which contribute to the output are activated. This results in a sparse network making it computationally efficient. It is also easier to implement this activation function in hardware and software since it does not involve any non-linear calculations. The ReLU activation function is provided in Figure 5.8.

5.2.3 Weight Set and Bias

The weight set of the neural network obtained after the training phase is stored in the form of a two dimensional integer array. The *int16* datatype is used for data representation of the weight set. It is also possible for the implementation to use different weight

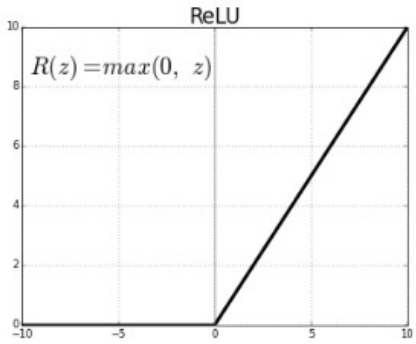

Figure 5.8: Rectified Linear Unit Activation Function

Network Architecture (a:b:c:d:e)	Total Number of Weights	Number of Multiplication operations	Number of Addition operations (including bias)
8:80:0:0:8	1280		1368
8:60:60:0:8	4560		4688
8:45:45:45:8	4770		4913

Table 5.4: Selection of Width and Depth parameter for S-Net

sets for each subsequent encryption. This allows the countermeasure to be resistant against possible templates that the adversary may create against the implementation.

5.2.4 Software Execution Flow

The software execution flow on the Pinata board is highlighted in Figure 5.9. Only one round of the encryption algorithm is executed as the *SubByte* function in the first round is the selected point of attack. The beginning and end of the round is indicated by trigger signals to the oscillopscope through one of the GPIO ports on the Pinata board. The plaintext to be encrypted is sent to the Pinata board via UART from the host computer and the corresponding ciphertext after the first round is communicated back to the host computer.

5.3 Attack Resistance Results

This section highlights the performed power analysis techniques on the unprotected and protected implementations of the *SubByte* operation of AES. The security of the countermeasure is evaluated using Differential Power Analysis (DPA), Correlation Power Analysis (CPA), Multivariate Distribution and Deep Learning based Template attacks. The results of the security of the implementations against these attacks are illustrated in the following subsections.

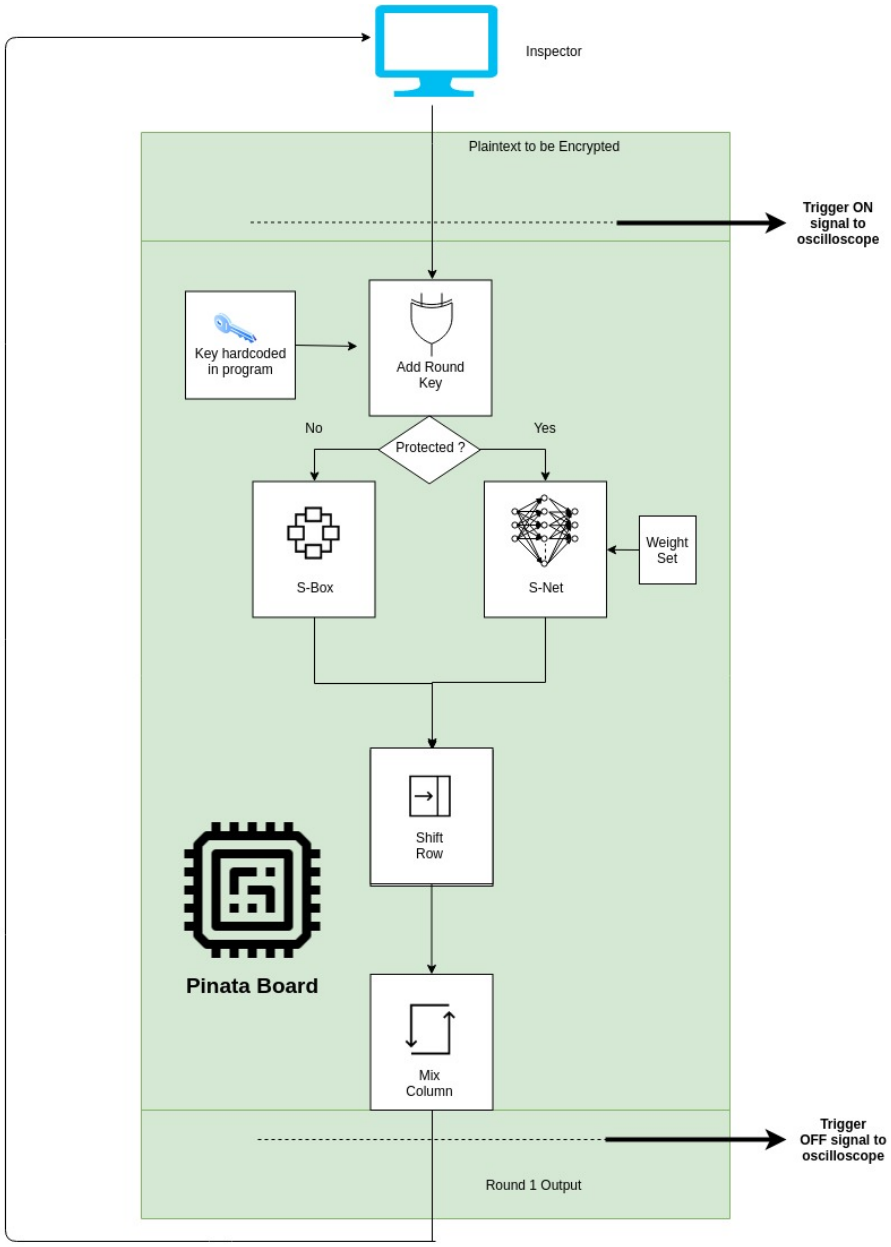

Figure 5.9: Software execution flow on Pinata for Protected and Unprotected AES

5.3.1 Security Against Differential Power Analysis

As discussed in Section 2.2.2.2, the DPA attacks identifies the correlation between the power signature and the intermediate value produced by the result of the *SubByte* operation by partitioning the traces. In this attack framework, the power consumed is quantized using the Hamming Weight model. Unlike the classical DPA, the attack used evaluates the correctness of the hypothesis on the intermediate value by partitioning according to the leakage model. The tracesets are partitioned into two groups based on

the value of the Hamming Weight of the intermediate value corresponding to the plaintext and the hypothesis on the key. The results of the attack are illustrated in Figure 5.10 and Figure 5.11. It can be seen that the difference in the differential coefficient for the unprotected implementation is large allowing easy identification of the correct hypothesis on the key. In case of the protected implementation, the correct SubKey does not appear within the top 100 ranks of possible SubKeys. This highlights the security of the S-Net. The evolution of the rank of the correct SubKey with the number of traces for the S-Net is illustrated in Figure 5.12.

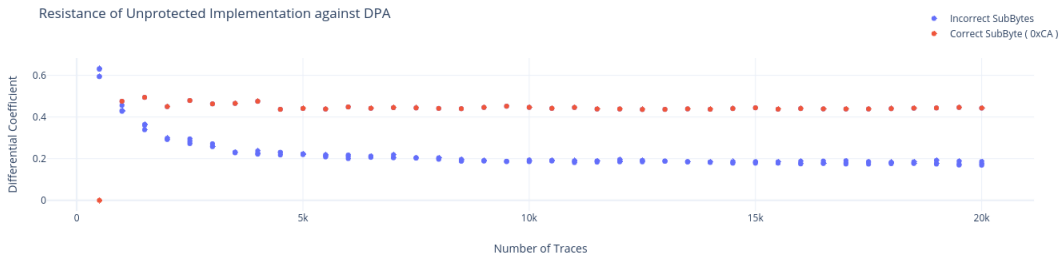

Figure 5.10: Security of Unprotected AES against DPA

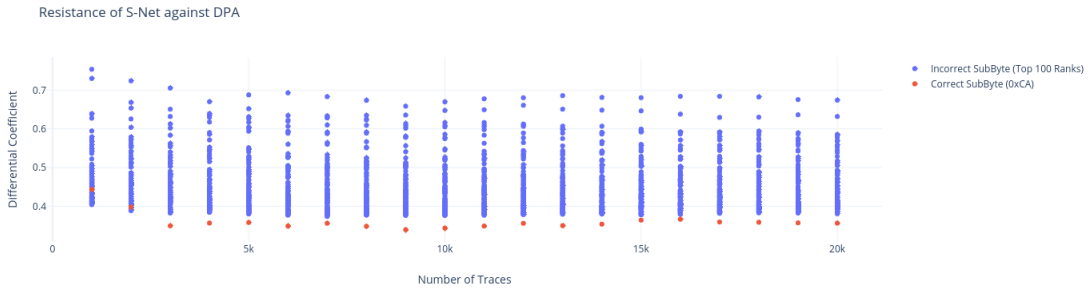

Figure 5.11: Security of S-Net against DPA

5.3.2 Security Against Correlation Power Analysis

As discussed in Section 2.2.2.3, the CPA attacks apply the correlation function on the trace matrix and the intermediate value. The Hamming Weight model is used in this attack to model the power consumed. The results obtained are similar to the DPA attack in the previous section. The correct SubKey (0XCA) does not appear within the top 100 ranks making the attack unsucessful against the protected implementation. This is illustrated in Figure 5.13 and Figure 5.14. The ranking analysis for this attack is illustrated in Figure 5.15.

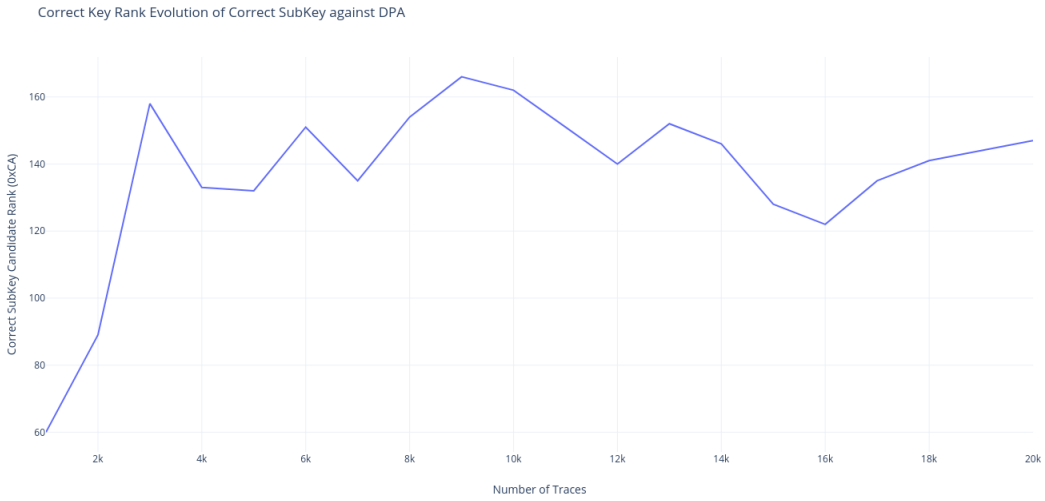

Figure 5.12: Ranking analysis of S-Net for DPA

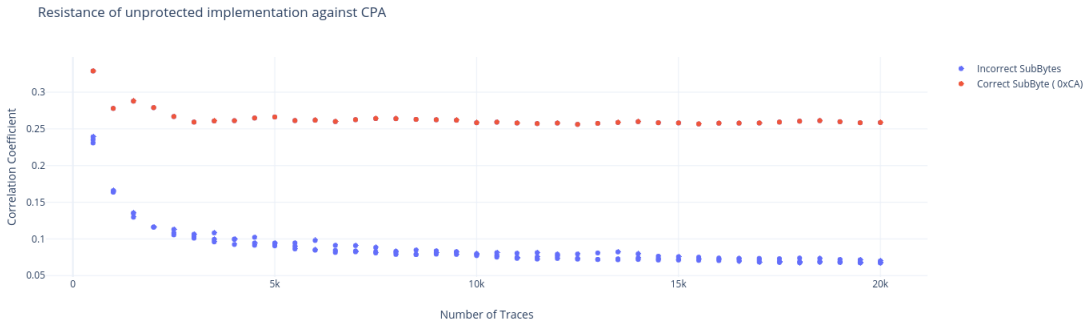

Figure 5.13: Security of Unprotected AES against CPA

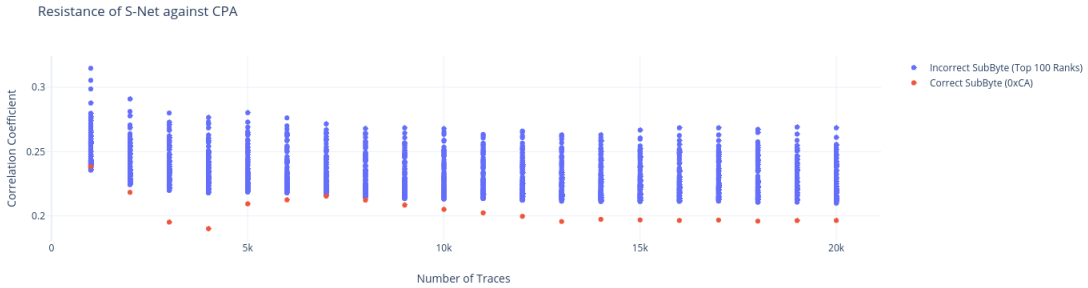

Figure 5.14: Security of S-Net against CPA

Figure 5.15: Ranking analysis of S-Net for CPA

5.3.3 Security Against Multivariate Distribution Template Attacks and Deep Learning Based Template Attacks

As discussed in Section 2.2.3.2 and 2.2.3.1, the profiled attacks involve the construction of a template in order to obtain the SubKey. Figure 5.16 and Figure 5.17 highlight the ranking evolution for the unprotected and S-Net implementations of AES for both attacks. The graphs indicate that it is not possible to obtain the SubKey in the case of the S-Net implementation. In case of the unprotected implementation, the correct SubKey achieves a high rank within 3000 traces for both the attacks.

5.3.4 Discussion

The unprotected implementation of AES and the S-Net was successfully implemented on software. It is observed that the correct SubKey can successfully be extracted for the unprotected implementation within 3000 traces for all attacks. The S-Net proves to be secure against all attacks. The security of the implementation was checked upto 20000 traces due to performance and memory constraints. None of the attacks were observed to be capable of successfully extracting the SubKey with a high degree of confidence.

The Hamming Weight leakage model is used to quantize the power consumed in all four attacks. The Hamming Distance (HD) model proves to be ineffective in attacking software. To successfuly use the Hamming Distance model, an in-depth knowledge of the mapping of the AES software on the computation and memory registers of the target implementation is required. Since the power consumption of the algorithm covers multiple clock cycles, it is an extremely strong requirement for the adversary to predict the position and value of the intermediate value in every instruction. Hence, the HD model is not effective in attacking software implementations of AES.

5.4 Performance Analysis

The performance of the unprotected implementation of AES is compared with the protected S-Net implementation. It is observed that the unprotected implementation requires an average execution time of 62.5 μs while the S-Net requires 4.7 ms for execution of one round of encryption. The execution time is calculated by activating a trigger at

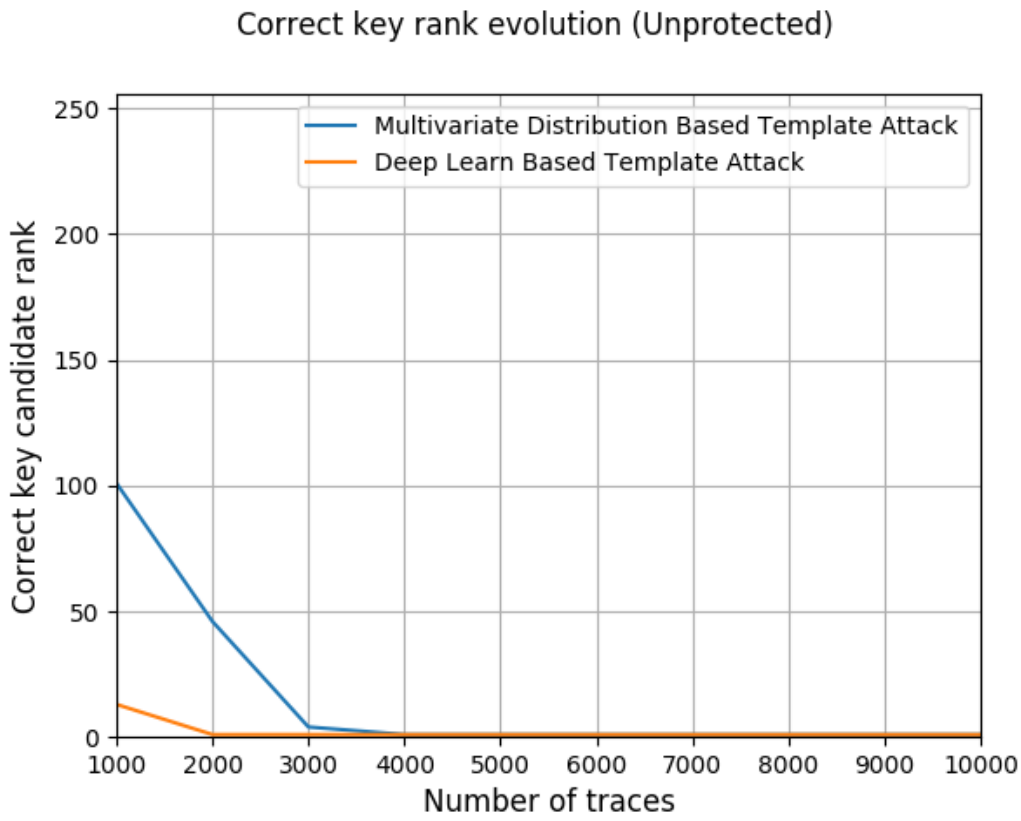

Figure 5.16: Ranking analysis of Unprotected AES against Profiled Attacks for correct SubByte

the start and end of the encryption on the Pinata board. It is concluded that the S-Net is 75 times slower than the unprotected implementation.

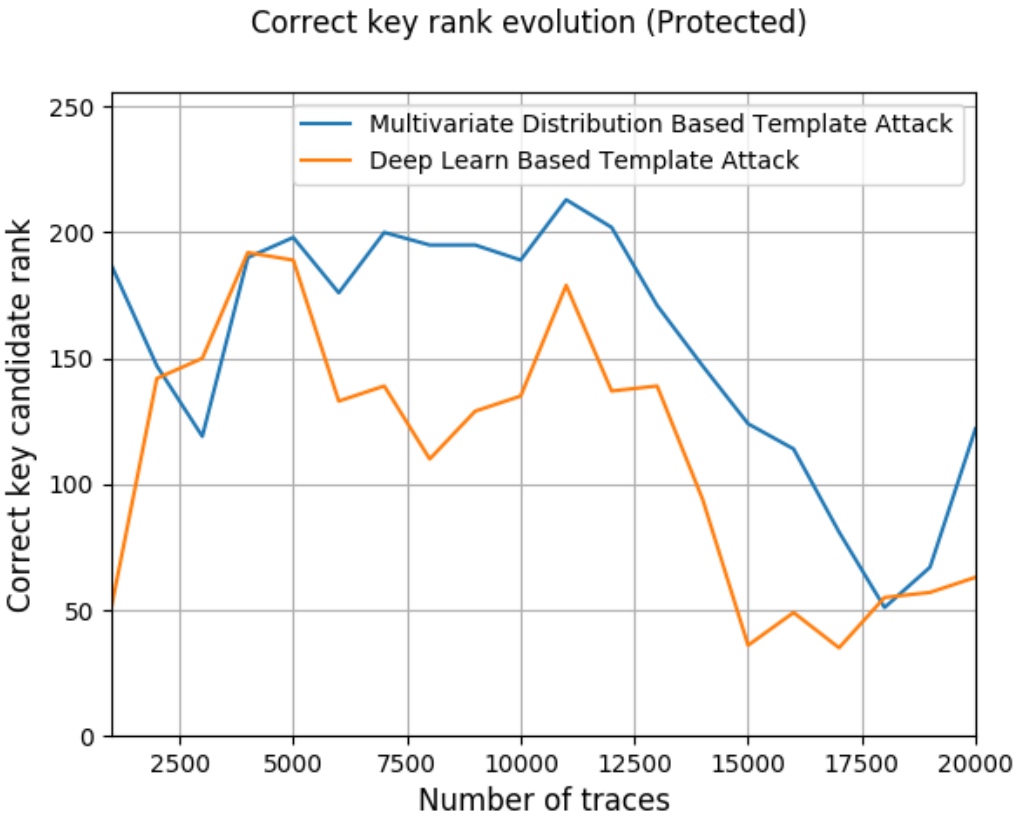

Figure 5.17: Ranking analysis of S-Net against Profiled Attacks for correct SubByte

6 Conclusion

This chapter provides a summary of the work included in this thesis and proposes future research work. Section 6.1 presents the summary of the conclusions of the dissertation. Section 6.2 provides the proposed future research work that may be implemented.

6.1 Summary

Chapter 1 of this thesis introduces the motivation behind the need for secure implementations of cryptographic algorithms. The chapter discusses the impact of side-channel analysis and the techniques used to secure devices against the same. It provides an introduction to the state-of-the-art countermeasures developed to protect devices against power based side-channel techniques.

Chapter 2 of this thesis gives a brief background of the targeted encryption algorithm, i.e, Advanced Encryption Standard (AES). Thereafter, it provides an overview of commonly employed power based side-channel analysis techniques. The state-of-the-art countermeasures against power based side-channel attacks are classified and illustrated with the help of examples.

Chapter 3 of this thesis gives a brief background of Artificial Neural Networks. Thereafter, the different hyperparameters that affect the performance of the neural network are classified and illustrated with the help of examples. An overview of the steps involved in the training and inference phase of the neural network is provided along with an illustration of the steps involved in the backpropagation algorithm .

Chapter 4 of this thesis provides the motivation behind usage of neural networks as a countermeasure. It explains the "breaking" of the leakage model using the S-Net. This is followed by the design methodology and optimization techniques implemented on the S-Net.

Chapter 5 of this thesis presents the experimental platform and the validation of the security of the implemented countermeasure. The chapter first highlights the experimental platform used to implement and validate the countermeasure. It provides a description of the implementation details of the S-Net in detail. Thereafter, it provides the results highlighting the degree of security of the implementation against various attacks. The results demonstrate that the S-Net is secure against all performed attacks. The thesis concludes that the implemented countermeasure effectively breaks the leakage model used by power analysis techniques. Finally, it provides a performance analysis of the implemented countermeasure.

6.2 Future Work

In this section, recommendations for future work that may be implemented for the topic addressed in the thesis is highlighted.

- **Optimized Weight Sets**. Weight sets containing multiples of two can be implemented using shift operations which can be easily implemented on hardware. The possibility of mapping the SubByte function using a neural network with binary weights can be further investigated.
- **Dedicated ASIC design for S-Net**. Fabrication of an ASIC capable of performing side-channel resistant encryption can be investigated.
- **Implementation of S-Net for other block ciphers**. There exists numerous encryption algorithms such as DES, PRESENT, CAESAR which involve the use of a substitution table in the encryption process. The possibility of implementing the S-Net in these encryption schemes must be investigated. Research on the security of the S-Net while mapping smaller substitution boxes is recommended.
- **Resistance of S-Net against other side-channel analysis techniques**. Research on the resistance of the implementation against temperature, electromagnetic and time-based attacks is highly recommended.
- **Memristor-based implementation**. Exploring the security and performance of memristor-based implementations of encryption algorithms is an excellent topic of research.

Bibliography

[1] C. Hegde, S. Manu, P. D. Shenoy, K. Venugopal, and L. Patnaik, "Secure authentication using image processing and visual cryptography for banking applications," in *2008 16th International Conference on Advanced Computing and Communications*. IEEE, 2008, pp. 65–72.

[2] S. G. Akl and P. D. Taylor, "Cryptographic solution to a problem of access control in a hierarchy," *ACM Transactions on Computer Systems (TOCS)*, vol. 1, no. 3, pp. 239–248, 1983.

[3] P. Hoffman and B. Schneier, "Attacks on cryptographic hashes in internet protocols," RFC 4270, November, Tech. Rep., 2005.

[4] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[5] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 45–60.

[6] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, and C. Sporleder, "Acoustic side-channel attacks on printers." in *USENIX Security symposium*, 2010, pp. 307–322.

[7] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em sidechannel (s)," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 29–45.

[8] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[9] T. S. Messerges, "Securing the aes finalists against power analysis attacks," in *International Workshop on Fast Software Encryption*. Springer, 2000, pp. 150–164.

[10] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Annual International Cryptology Conference*. Springer, 1999, pp. 398–412.

[11] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, 2011.

[12] M.-L. Akkar and C. Giraud, "An implementation of des and aes, secure against some attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2001, pp. 309–318.

[13] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A side-channel analysis resistant description of the aes s-box," in *International workshop on fast software encryption*. Springer, 2005, pp. 413–423.

[14] M. Rivain and E. Prouff, "Provably secure higher-order masking of aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 413–427.

[15] K. Tiri, M. Akmal, and I. Verbauwhede, "A dynamic and differential cmos logic with signal independent power consumption to withstand differential power analysis on smart cards," in *Proceedings of the 28th European solid-state circuits conference*. IEEE, 2002, pp. 403–406.

[16] J. A. Ambrose, S. Parameswaran, and A. Ignjatovic, "Mute-aes: A multiprocessor architecture to prevent power analysis based side channel attack of the aes algorithm," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2008, pp. 678–684.

[17] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[18] A. Biryukov and D. Khovratovich, "Related-key cryptanalysis of the full aes-192 and aes-256," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2009, pp. 1–18.

[19] S. Guilley, P. Hoogvorst, and R. Pacalet, "Differential power analysis model and some results," in *Smart Card Research and Advanced Applications Vi*. Springer, 2004, pp. 127–142.

[20] R. X. Gu and M. I. Elmasry, "Power dissipation analysis and optimization of deep submicron cmos digital circuits," *IEEE Journal of solid-state circuits*, vol. 31, no. 5, pp. 707–713, 1996.

[21] L. Lerman, G. Bontempi, and O. Markowitch, "A machine learning approach against a masked aes," *Journal of Cryptographic Engineering*, vol. 5, no. 2, pp. 123–139, 2015.

[22] J. Heyszl, A. Ibing, S. Mangard, F. De Santis, and G. Sigl, "Clustering algorithms for non-profiled single-execution attacks on exponentiations," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2013, pp. 79–93.

[23] R. Mayer-Sommer, "Smartly analyzing the simplicity and the power of simple power analysis on smartcards," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 78–92.

[24] O. Lo, W. J. Buchanan, and D. Carson, "Power analysis attacks on the aes-128 s-box using differential power analysis (dpa) and correlation power analysis (cpa)," *Journal of Cyber Security Technology*, vol. 1, no. 2, pp. 88–107, 2017.

[25] M. F. Witteman, J. G. van Woudenberg, and F. Menarini, "Defeating rsa multiply-always and message blinding countermeasures," in *Cryptographers Track at the RSA Conference*. Springer, 2011, pp. 77–88.

[26] P. Kocher, J. Jaffe, B. Jun *et al.*, "Introduction to differential power analysis and related attacks," 1998.

[27] S. Mangard, "A simple power-analysis (spa) attack on implementations of the aes key expansion," in *International Conference on Information Security and Cryptology*. Springer, 2002, pp. 343–358.

[28] B. Gierlichs, "Introduction to power analysis," ECRYPT Summer School, 2011.

[29] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.

[30] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers," in *European Symposium on Research in Computer Security*. Springer, 1998, pp. 97–110.

[31] B. Gierlichs, L. Batina, B. Preneel, and I. Verbauwhede, "Revisiting higher-order dpa attacks," in *Cryptographers Track at the RSA Conference*. Springer, 2010, pp. 221–234.

[32] T. S. Messerges, "Using second-order power analysis to attack dpa resistant software," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 238–251.

[33] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert, "Soft analytical side-channel attacks," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 282–296.

[34] C. Archambeau, E. Peeters, F.-X. Standaert, and J.-J. Quisquater, "Template attacks in principal subspaces," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 1–14.

[35] J. D. Golić and C. Tymen, "Multiplicative masking and power analysis of aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 198–212.

[36] D. Suzuki and M. Saeki, "Security evaluation of dpa countermeasures using dual-rail pre-charge logic style," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 255–269.

[37] W. Luis, G. R. Newell, and K. Alexander, "Differential power analysis countermeasures for the configuration of sram fpgas," in *MILCOM 2015-2015 IEEE Military Communications Conference*. IEEE, 2015, pp. 1276–1283.

[38] Q. Tian, A. Shoufan, M. Stoettinger, and S. A. Huss, "Power trace alignment for cryptosystems featuring random frequency countermeasures," in *2012 Second International Conference on Digital Information Processing and Communications (ICDIPC)*. IEEE, 2012, pp. 51–55.

[39] Z. Chen and Y. Zhou, "Dual-rail random switching logic: a countermeasure to reduce side channel leakage," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 242–254.

[40] M. Tunstall and O. Benoit, "Efficient use of random delays in embedded software," in *IFIP International Workshop on Information Security Theory and Practices*. Springer, 2007, pp. 27–38.

[41] Y. Lu, M. P. O'Neill, and J. V. McCanny, "Fpga implementation and analysis of random delay insertion countermeasure against dpa," in *2008 International Conference on Field-Programmable Technology*. IEEE, 2008, pp. 201–208.

[42] J. A. Ambrose, R. G. Ragel, and S. Parameswaran, "Rijid: random code injection to mask power analysis based side channel attacks," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 489–492.

[43] Y. Zafar, J. Park, and D. Har, "Random clocking induced dpa attack immunity in fpgas," in *2010 IEEE International Conference on Industrial Technology*. IEEE, 2010, pp. 1068–1070.

[44] H. J. Mahanta, A. K. Azad, and A. K. Khan, "Differential power analysis: Attacks and resisting techniques," in *Information Systems Design and Intelligent Applications*. Springer, 2015, pp. 349–358.

[45] P.-C. Liu, H.-C. Chang, and C.-Y. Lee, "A low overhead dpa countermeasure circuit based on ring oscillators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 7, pp. 546–550, 2010.

[46] N. Kamoun, L. Bossuet, and A. Ghazel, "Correlated power noise generator as a low cost dpa countermeasures to secure hardware aes cipher," in *2009 3rd International Conference on Signals, Circuits and Systems (SCS)*. IEEE, 2009, pp. 1–6.

[47] Y. Zafar and D. Har, "A novel countermeasure enhancing side channel immunity in fpgas," in *2008 International Conference on Advances in Electronics and Microelectronics*. IEEE, 2008, pp. 132–137.

[48] P. Yu, "Implementation of dpa-resistant circuit for fpga," Ph.D. dissertation, Virginia Tech, 2007.

[49] L. Goubin and A. Martinelli, "Protecting aes with shamirs secret sharing scheme," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 79–94.

[50] L. Deng, M. Aksmanovic, X. Sun, and C. J. Wu, "Speech recognition using hidden markov models with polynomial regression functions as nonstationary states," *IEEE transactions on speech and audio processing*, vol. 2, no. 4, pp. 507–520, 1994.

[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[52] C. Chen, S. Duan, T. Cai, and B. Liu, "Online 24-h solar power forecasting based on weather type classification using artificial neural network," *Solar energy*, vol. 85, no. 11, pp. 2856–2870, 2011.

[53] J. Goh, S. Adepu, M. Tan, and Z. S. Lee, "Anomaly detection in cyber physical systems using recurrent neural networks," in *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, 2017, pp. 140–145.

[54] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on neural networks*, vol. 3, no. 5, pp. 683–697, 1992.

[55] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[56] J. C. B. Gamboa, "Deep learning for time-series analysis," *arXiv preprint arXiv:1701.01887*, 2017.

[57] A. Moujahid, "A practical introduction to deep learning with caffe and python," 2016.

[58] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[59] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[60] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[61] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

[62] Z. Xu, H. Zhang, Y. Wang, X. Chang, and Y. Liang, "L 1/2 regularization," *Science China Information Sciences*, vol. 53, no. 6, pp. 1159–1169, 2010.

[63] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *arXiv preprint arXiv:1611.03530*, 2016.

[64] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[65] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*, 2016.

[66] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Examining smart-card security under the threat of power analysis attacks," *IEEE transactions on computers*, vol. 51, no. 5, pp. 541–552, 2002.

[67] D. Cournapeau *et al.*, "Scikit learn: Open source machine learning library for Python," 2007. [Online]. Available: https://en.wikipedia.org/wiki/Scikit-learn

[68] F. Chollet *et al.*, "Keras: Open source neural network library for Python," 2015. [Online]. Available: https://en.wikipedia.org/wiki/Keras

[69] Nicolas Limare, Integer and Floating Point Performance, 2014, [Online; accessed December 27, 2018]. [Online]. Available: http://nicolas.limare.net/pro/notes/2014/12/12_arit_speed/

[70] Pico Technology, Oscilloscopes, 2018, [Online; accessed December 27, 2018]. [Online]. Available: https://www.picotech.com/images/uploads/threesixty/3400/picoscope_3400_35.jpg

[71] Riscure, Software Training Target, 2018, [Online; accessed December 27, 2018]. [Online]. Available: https://www.riscure.com/product/pinata-training-target/

[72] Riscure, Current Probe, 2018, [Online; accessed December 27, 2018]. [Online]. Available: https://www.riscure.com/product/current-probe/

[73] Riscure, Security Tools-Inspector Side Channel Analysis, 2018, [Online; accessed December 27, 2018]. [Online]. Available: https://www.riscure.com/security-tools/inspector-sca/

[74] K. Valk *et al.*, "trsfile: Open source trs file support tools for Python," 2019. [Online]. Available: https://pypi.org/project/trsfile/