

**DESIGN AND IMPLEMENTATION OF A PORTABLE
HARDWARE DEVICE FOR ANALYZE SIDE
CHANNEL ATTACKS**

Theshan Pasindu Wijesinghe

IT21325328

Bsc (Hons) Degree In Information Technology
Specializing In Cyber Security

Department Of Computer System Engineering


Sri Lanka Institute of Information Technology
Sri Lanka

April 2025


DECLARATION

I declare that this is my own work, and this dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology the non-exclusive right to reproduce and distribute my dissertation in whole or part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: 

Date: 09/04/2025

Signature of the Supervisor: 

Date: 09/04/2025

ABSTRACT

Side-channel attacks (SCAs) pose a significant threat to the security of cryptographic implementations, as they exploit physical leakages such as electromagnetic (EM) emissions rather than attempting to break encryption algorithms mathematically. This research focuses on developing a low-cost, portable hardware-based system capable of detecting EM-based side-channel vulnerabilities in embedded cryptographic devices.

The core objective of this research was to analyze EM traces associated with encryption processes and identify patterns that may indicate potential vulnerabilities. A machine learning-based approach was adopted, where multiple classifiers Random Forest, SVM, KNN, and Logistic Regression were evaluated based on performance metrics such as accuracy, precision, recall, and F1-score. Among them, the Random Forest model demonstrated the best balance between accuracy and computational efficiency, achieving over 80% classification accuracy with minimal preprocessing.

The trained model was deployed on a Raspberry Pi 4, transforming the system into a fully functional and self-contained detection tool. The Raspberry Pi was configured to accept user input and analyze EM traces, outputting the prediction either "Vulnerable" or "Non-Vulnerable" on an attached 16x2 LCD screen. The system was designed to run efficiently without the need for high-end hardware, making it suitable for academic researchers and university students working in constrained environments.

Results show that simple statistical features, such as the mean of EM traces, can effectively guide model training while preserving performance. The system proved to be accurate, responsive, and robust under hardware limitations, demonstrating that machine learning can be reliably used for side-channel vulnerability detection in real-world applications. This work contributes a practical and accessible tool for early-stage cryptographic hardware evaluation, with potential extensions in educational and research contexts.

ACKNOWLEDGEMENT

I would like to thank Mr. Kavinga Yapa my supervisor, first for his unfaltering guidance, support and encouragement during the course of this research project. His expertise, valuable comments and tireless motivation helped in steering the course of my work and negotiating difficulties at every stage.

I am also very grateful to my co-supervisor, Mr. Amila Senaratne for his creative suggestions, technical guidance and continuous support. His contributions were a key factor in helping me to formalize my thoughts and translate theoretical principles into practical applications.

Finally, I would like to express my gratitude to the academic and technical staff of the department for creating a conducive and intellectually stimulating environment. I would like to express my special thanks to my fellow students and colleagues who have provided inspiration, support, and friendship throughout this journey.

This project would not have been possible without the contributions of all these individuals, and I remain sincerely thankful to each of them.

TABLE OF CONTENT

DECLARATION	i
ABSTRACT	ii
ACKNOWLEDGEMENT	iii
TABLE OF CONTENT	iv
LIST OF FIGURES	vii
LIST OF ABBREVIATIONS	viii
1. INTRODUCTION	1
1.1 Background Literature	1
1.1.1 Side-Channel Attacks and Their Significance in Cryptographic Security	1
1.1.2 Electromagnetic Side-Channel Analysis	1
1.1.3 Machine Learning-Based Side-Channel Analysis	2
1.1.4 Existing Research and Real-World Applications	2
1.2 Research Gap	3
1.3 Research Problem	3
1.3.1 Challenges in Existing EM Vulnerability Detection	4
1.3.2 Defining the Research Problem	4
1.4 Why This Research Is Important	6
1.5 Research Objective	6
2. METHODOLOGY	9
2.1 Data Gathering	11
2.1.1 Arduino-Based Cryptographic Environment	11
2.1.2 Oscilloscope Setup for EM Trace Capture	12

2.1.3	Structuring the Dataset	12
2.1.4	Alternative EM Trace Collection Attempt Using Copper Shielded Ethernet Setup	14
2.2	Preprocessing and Feature Extraction	16
2.2.1	Importance of Preprocessing in Side-Channel Analysis	16
2.2.2	Cleaning and Denoising EM Traces	17
2.2.3	Normalization and Scaling	19
2.2.4	Feature Engineering	21
2.2.5	Labeling the Dataset	24
2.2.6	Structuring the Preprocessed Dataset	29
2.3	Machine Learning Model Training	32
2.3.1	Model Selection	33
2.3.2	Final Model Selection and Evaluation	36
2.4	Implementing on Hardware (Raspberry Pi Deployment)	38
2.4.1	Laptop-Based Setup	38
2.4.2	ESP32 and Other Microcontroller Boards	40
2.4.3	NVIDIA Jetson Nano	41
2.4.4	Final Selection – Raspberry Pi 4	43
2.4.5	Raspberry Pi Version 4 Model Specifications	44
2.4.6	System Setup and Deployment Workflow	46
2.5	Evaluation and Testing	50
3.	COMMERCIALIZATION ASPECTS OF THE PRODUCT	52
4.	RESULTS AND DISCUSSIONS	53
4.1	Results	53

4.1.1	Machine Learning Model Results	53
4.1.2	Final Output Results	55
4.2	Research Findings	56
4.3	Discussion	57
5.	Conclusion	59
	References	62

LIST OF FIGURES

Figure 1: Correlation Matrix54

Figure 2: Model Accuracy Comparison Graph55

Figure 3: Final Output Results56

LIST OF ABBREVIATIONS

Abbreviation	Description
SCA	Side-channel attacks
EM	Electromagnetic
SVM	Support Vector Machine
FFT	Fast Fourier Transform
DEMA	Differential Electromagnetic Analysis
CSV	Comma-Separated Values
KNN	K-Nearest Neighbors
GPIO	General Purpose Input/Output
GUI	Graphical User Interface

1. INTRODUCTION

1.1 Background Literature

1.1.1 Side-Channel Attacks and Their Significance in Cryptographic Security

Existing cryptographic systems are rendered mathematically secure but physically insecure to side-channel attacks (SCAs). SCAs exploit unintentional information leakage throughout cryptographic processing, e.g., power consumption fluctuations, electromagnetic (EM) radiation, and run time. SCAs differ from traditional cryptanalysis, which attempts to break cryptographic algorithms based on mathematical weaknesses, in that they aim at recovering encryption keys from physical measurements of the device performing the encryption operation.

Among the various types of side-channel attacks, EM side-channel analysis is particularly concerning because it is a non-invasive attack that does not require direct access to internal components. EM leakage occurs as a result of current fluctuations within a device's circuitry, producing electromagnetic waves that can be captured and analyzed. Adversaries can use this information to infer encryption keys, making even well-established encryption algorithms vulnerable.

1.1.2 Electromagnetic Side-Channel Analysis

Electromagnetic attacks exploit the fact that electronic devices emit various electromagnetic signals when they are functioning. When a cryptographic operation such as AES or DES is being executed, these signals can be observed to extract secret information. Correlation Power Analysis and Differential Power Analysis attacks have been used with success to extract cryptographic keys by correlating EM traces with theoretical models of power consumption.

Prior research has shown that cryptographic hardware, such as microcontrollers and embedded systems, have unique EM leakage patterns based on the encryption key employed. This has created severe issues in secure hardware design, and as a result, countermeasures like hardware shielding, noise injection, and randomized execution

patterns have been designed. These methods do not completely remove EM leakage, and vulnerability detection is thus a significant research area.

1.1.3 Machine Learning-Based Side-Channel Analysis

Machine learning techniques have, in the recent years, emerged as efficient tools for side-channel analysis. Statistical relationship and differential computation are employed in the traditional SCA techniques, whereas machine learning techniques can learn complex patterns in side-channel signals without necessarily knowing the exact mathematical representations of the cryptographic implementation. Supervised learning techniques like Random Forest, SVM, and Neural Networks have been used successfully to classify EM traces as vulnerable or non-vulnerable.

One of the key benefits of machine learning for side-channel analysis is that it can be trained to accept various encryption implementations and levels of noise. By training on known EM traces, researchers can create classifiers that can identify vulnerabilities in new cryptographic devices. Despite these breakthroughs, however, much of the current research depends on high-end lab equipment, which makes machine learning-based side-channel detection in the real world impractical.

1.1.4 Existing Research and Real-World Applications

A number of research studies attempted to find and detect EM side-channel vulnerability. Such research studies, nevertheless, depend upon the use of costly oscilloscopes and specialized hardware and rest are impractical for security analysis on an economical basis. Furthermore, cryptographic devices in practice, including bank terminal embedded systems and IoT equipment, are vulnerable to side-channel attacks due to the fact that they lack live vulnerability detection software.

To overcome this problem, researchers have investigated low-cost implementations of side-channel detection, but they are typically consisting with poor portability and usability. This research proposes to fill this gap by presenting a hardware-based detection system that incorporates machine learning models into a low-cost Raspberry Pi, which is an affordable EM side-channel vulnerability detection tool.

1.2 Research Gap

In spite of the great advancements in side-channel analysis, there are some critical gaps in current research. To begin with, most of the current research targets cryptographic implementations instead of determining weaknesses first. Developers and security experts need proactive vulnerability checking software to examine encryption processes to see if there are weaknesses.

Second, existing detection methods rely on sophisticated laboratory equipment involving costly oscilloscopes and sophisticated signal processing techniques. The methods are inappropriate for use in the real world, especially in resource-constrained environments where cost and availability matter. A practical solution must be low-cost yet accurate, and it must permit security analysis on widely available embedded platforms.

Third, while machine learning has been promising in side-channel analysis, most implementations are computationally intensive and need large training sets. The challenge is in using machine learning models in low-power embedded systems and efficient classification of EM traces.

This work bridges this gap by creating a low-cost, portable hardware platform capable of detecting EM vulnerabilities through machine learning. The platform, based on a Raspberry Pi and an oscilloscope, provides an affordable and viable method of cryptographic security testing on actual devices.

1.3 Research Problem

Cryptographic security is a key part of ensuring confidentiality and integrity of information in modern digital systems. Secure communication, financial transactions, and national security uses are only a few examples of applications in which encryption forms the basis of cybersecurity. However, even though cryptographic algorithms such as AES and DES are mathematically secure, they are still vulnerable to physical attacks known as side-channel attacks (SCAs). SCAs exploit incidental information leakage such as power consumption, execution time, and electromagnetic (EM) emissions during encryption and

decryption processes to acquire secret keys without ever compromising the cryptographic algorithm itself.

Among many side-channel attacks, Electromagnetic Side-Channel Attacks are particularly concerning because they are non-invasive. By simply monitoring the electromagnetic radiation of a crypto device during encryption, an attacker can deduce secret keys with high accuracy. The attack is particularly problematic for embedded systems, IoT, and critical infrastructure where encryption is prevalent but frequently lacks proper physical security countermeasures.

1.3.1 Challenges in Existing EM Vulnerability Detection

Several countermeasures have been proposed to defend against EM-SCAs, including hardware shielding, execution time randomization, noise injection, and masking. These countermeasures are not necessarily effective, however, as they have a tendency to make performance, cost, and power efficiency trade-offs. Furthermore, most cryptographic hardware continues to leak EM signals that can be detected, and therefore they are still susceptible to attack even with protections in place.

One of the biggest challenges to safeguarding cryptographic hardware against EM-SCAs is the lack of low-cost and affordable vulnerability assessment tools. Detection is currently carried out with the use of costly oscilloscopes, advanced signal processing, and professional-level analysis, which are not affordable for small organizations, researchers, and solo security researchers. There is, therefore, an urgent need for a low-cost and portable device that can automatically analyze and detect EM side-channel vulnerabilities in cryptographic implementations without advanced technical expertise.

1.3.2 Defining the Research Problem

With the shifting paradigm of cyber-attacks, particularly on hardware-based implementations, the need for efficient and cost-effective ways for vulnerability detection has become ever so important. Even though cryptographic algorithms are securely mathematically, they are generally implemented on physical platforms that emit unintentional signals such as electromagnetic (EM) radiation upon use. Such emanations

leak information, making devices vulnerable to side-channel attacks. While the problem is of a serious nature, most of the detection approaches that exist today are either highly costly, technologically demanding, or suitable for high-end lab environments. This work thus seeks to address an age-old question:

"How can a portable and cost-effective hardware-based system be developed to detect electromagnetic side-channel vulnerabilities in cryptographic implementations using machine learning?"

To solve this problem, the research is focused on the design and development of a low-cost real-time system capable of analyzing EM traces and determining if a target cryptographic process is vulnerable to EM-based side-channel attacks. The method begins with the setup of a cryptographic environment using an Arduino board. EM emissions from the Arduino during encryption operations are observed using an oscilloscope. The signals are captured and saved in a dataset with corresponding plaintext and encryption key inputs to allow for detailed analysis of side-channel leakage behavior.

The second step involves the application of machine learning techniques for automation of vulnerability detection. A labeled dataset of EM traces is employed to train a supervised learning model based on trace features and statistical metrics. Mean values, signal amplitude, regularity of the waveform, and similar features typical of leakage in cryptography are some of these features. Based on this labeled dataset, the model learns to identify encryption traces as vulnerable or non-vulnerable based on the type of EM radiation.

To guarantee the practicability and usability of the solution, the trained model is implemented on a Raspberry Pi 4, a light-weight, portable analysis platform. The Raspberry Pi is programmed to accept encryption input parameters (the key and the plaintext), process the corresponding EM trace, and output the vulnerability classification result on an available LCD display. This hardware-based platform converts sophisticated detection methodology into a user-friendly tool irrespective of specialized technical knowledge and high-end lab equipment.

Finally, the system is used to perform real-time analysis, providing instant feedback regarding the state of vulnerability of encryption operations. Developers, security researchers, and organizations can now include side-channel detection in the development and test phases of their cryptographic systems. This way, vulnerabilities can be identified early on, and preventive measures can be undertaken before the attackers can exploit them. This research thus provides a cost-effective and scalable solution for the growing need to make hardware-based cryptographic implementations secure against electromagnetic side-channel attacks.

1.4 Why This Research Is Important

This research is important since the majority of current research in EM side-channel attacks is concentrated on offensive tactics showing attacks and not on countermeasure or detection tool development. Although many attack methods have been reported, few solutions are readily available for proactively detecting and preventing EM vulnerabilities prior to an attack.

In addition, as IoT devices, embedded systems, and edge computing devices become more common, their security against EM side-channel attacks is becoming more and more critical. Most of these devices are used in environments where physical security cannot always be assured, so they are the most likely targets for attackers who want to obtain cryptographic keys from electromagnetic leakage.

By offering a low-cost, portable, and machine learning-based vulnerability detection tool, this work provides a scalable and practical solution for securing cryptographic implementations. The results of this research may assist device manufacturers, security professionals, and researchers in making encryption hardware more resilient to actual side-channel attacks.

1.5 Research Objective

The main goal of this research is to create a hardware-based system that can identify electromagnetic side-channel vulnerabilities in cryptographic implementations. Since side-channel attacks have become a major threat to secure encryption systems, this

research will fill the gap by providing an inexpensive, portable, and machine learning-driven solution for EM vulnerability detection. In order to achieve this overall objective, the research concentrates on a number of important objectives that, in aggregate, contribute to the design, implementation, and evaluation of the suggested system.

One of the core objectives is to develop a hardware configuration for the acquisition of EM traces during the process of encryption. This configuration comprises an Arduino board, functioning as the cryptographic platform for carrying out encryption operations, and an oscilloscope, employed to monitor and record electromagnetic emissions produced in these operations. The Arduino board implements pre-defined encryption algorithms, while the oscilloscope captures EM traces released by the board. The hardware-based solution provides real-world utility, rendering the system appropriately suitable for cryptographic device analysis under real-world conditions.

Once the raw EM traces have been captured, the subsequent task is to process and analyze gathered data to make the system more accurate and trustworthy. Electromagnetic signals include noise and irregular variations, and these can distort meaningful patterns. To counteract this, methods of noise reduction and signal normalization are used in the EM traces prior to the analysis. By preprocessing the data, the system ensures extraction of only relevant features, thereby resulting in a more efficient classification process.

One of the main emphases in this research is to develop a machine learning model that can identify vulnerable and non-vulnerable encryption processes using the EM traces. A supervised learning paradigm is used, wherein the model is trained using a labeled dataset with pre-classified EM trace samples. By discovering the underlying patterns in the EM emissions, the model is able to make educated predictions regarding whether a specific encryption process is vulnerable to EM side-channel attacks. This automated classification avoids the need for manual analysis, thus making the system more efficient and accessible to researchers. Once trained, the following goal is to deploy the trained machine learning model on a Raspberry Pi and make the system portable and an economical real-time vulnerability detection solution.

The Raspberry Pi acts as the central computation module, executing the trained model and examining acquired EM traces. The system is designed to be user-friendly, with an LCD display providing real-time feedback on whether the observed encryption process is vulnerable or secure. This implementation ensures that the detection mechanism is lightweight, scalable, and practical for real-world applications. Lastly, the research will assess the performance of the constructed system regarding classification accuracy, computational complexity, and usability. The system is evaluated on various encryption scenarios to gauge its efficiency in detecting EM vulnerabilities. By a comprehensive analysis of its capability, this research makes sure that the system is not only theoretically robust but also practically feasible for researchers and university students who seek to protect their cryptographic implementations against side-channel attacks.

By reaching these goals, this research helps advance the study of cryptographic security through the innovation of a new, machine learning-based method of EM side-channel vulnerability detection.

The outcomes of this work have the promise to make it easier to make embedded systems, IoT devices, and other cryptography uses more secure, allowing companies to be ahead of the threat of advanced side-channel attacks. This research is a progression towards creating implementable, economically viable, and automated methods to protect encryption devices from emerging threats to cybersecurity.

2. METHODOLOGY

Creating a hardware-based system for identifying electromagnetic (EM) side-channel vulnerabilities in cryptographic implementations is a methodical and systematic process. The approach used in this study is to systematically gather EM traces, process the gathered data, train a machine learning model, implement the trained model on a hardware platform, and test its performance. The whole process ensures that the system can identify cryptographic vulnerabilities accurately while being efficient and portable.

Step one is to create a controlled environment for encryption with cryptographic operations on a hardware system. This allows us to tap the unwanted electromagnetic emissions during the process of encryption and study them to see if they are susceptible to side-channel attacks. EM traces that have been gathered go through preprocessing where noise is eliminated and features relevant to classification are extracted. After preparing the data, a machine learning model is trained through supervised learning methods so that it can distinguish between vulnerable and non-vulnerable encryption traces. The trained model is then deployed on a Raspberry Pi to develop a portable detection system. Lastly, the system is tested and analyzed to assess its performance in real-world scenarios.

This methodology can be divided into five key stages:

- **Data Gathering** – The initial step involves setting up an encryption platform with an Arduino board in order to perform cryptographic functions. When encrypting, electromagnetic emissions are recorded through the use of an oscilloscope. The emissions change depending on the process of encryption and are stored as a dataset with relevant plaintext inputs and encryption keys.
- **Preprocessing and Feature Extraction** – The acquired EM traces will be noisy and contain irrelevant variations, which require processing prior to analysis. Techniques of filtering are used to eliminate unwanted signals, and pertinent statistical features (including mean values) are extracted. This process will ensure

that the dataset employed to train the machine learning model consists of meaningful data that can separate vulnerable and non-vulnerable traces.

- **Machine Learning Model Training** – Following the preprocessing of the dataset, a supervised learning model is trained to categorize encryption processes according to their EM trace patterns. The dataset is marked in terms of trace features, and the model is trained to learn the differences between secure and vulnerable instances of encryption. This is an essential step in enabling the system to automatically detect possible vulnerabilities without the need for human intervention.
- **Implementing on Hardware (Raspberry Pi Deployment)** – After the machine learning model is trained and tested, it is implemented onto a Raspberry Pi. The Raspberry Pi is configured to accept encryption inputs (plaintext and key), process the related EM traces, and output the classification result on an LCD display. This deployment turns the research into a working, portable hardware system that can be utilized for real-time vulnerability detection.
- **Evaluation and Testing** – The last phase of the methodology is evaluating the performance of the system based on classification accuracy, computational performance, and real-world applicability. The accuracy of the model is validated with several datasets, and the system is tested in varying conditions to provide reliability. Performance metrics are evaluated to check the effectiveness of the detection system to detect electromagnetic side-channel vulnerabilities.

By adhering to this systematic approach, this study formulates a low-cost, practical solution for the detection of cryptographic vulnerabilities due to electromagnetic emissions. Every phase is properly tailored so that the resulting system is accurate, effective, and usable in real-world security audits.

2.1 Data Gathering

The data collection phase is the building block step of this research, as the performance of the machine learning model greatly relies on the quality and relevance of the collected data. The aim of this phase was to establish a controlled and reproducible environment that would be able to mimic real-world cryptographic operations and produce electromagnetic (EM) emissions during these operations. These emissions were then recorded, organized, and utilized to construct a dataset appropriate for machine learning-based vulnerability analysis.

2.1.1 Arduino-Based Cryptographic Environment

For the purpose of simulating encryption processes, an Arduino Uno microcontroller was employed as the central device tasked with performing cryptography procedures. The Arduino Uno is a low-cost and popular microcontroller board that implements the ATmega328P processor. It can support simple digital I/O functions, thus being apt for simple encryption processes and perfect for test runs in a resource-limited embedded system.

The Arduino was coded with the Arduino IDE and an embedded preset encryption function in this example, a minimal version of DES or a reduced custom encryption function to mimic real-time data processing. The code took input in the form of a plaintext and a 16-character key, executed the encryption processes, and printed the ciphertext. While the encryption function was running, the internal data processing activity of the microcontroller produced changing electrical currents. These fluctuations caused EM emissions from the board's circuitry, especially around the CPU and power lines.

To provide a controlled and repetitive environment, the Arduino was driven via USB by a laptop and all unwanted peripherals were shut down to reduce noise. The cryptographic operations were run repeatedly using varied plaintext-key pairs to mimic various operational conditions and create a broad spectrum of EM emissions for examination.

2.1.2 Oscilloscope Setup for EM Trace Capture

In order to record the electromagnetic signals radiated by the Arduino while encrypting, an oscilloscope in the university lab was utilized. Oscilloscopes can measure voltage changes over time, and when combined with suitable probes, they can be utilized to measure electromagnetic radiation from electronic circuits.

Since there were no high-end near-field EM probes available as the budget for this project was a concern, a workaround was used. The oscilloscope was coupled to the Arduino via normal passive probes, and the measurement was targeted at most likely to leak EM signals such as the VCC (power supply pin), GND (ground), and regions near the microcontroller's clock line or processing core. In a few instances, a small wire loop or coil was employed as an ad hoc near-field probe to capture radiated EM signals, held very close to the surface of the chip. The wire loop was connected to the channel input of the oscilloscope using a BNC connector for signal acquisition.

The oscilloscope was configured with the following general settings:

- **Sample rate:** High enough (e.g., 1 MS/s or more) to capture fine-grained variations in EM emissions.
- **Time base:** Adjusted to display the entire duration of the encryption function.
- **Trigger:** Set to begin capturing when a voltage spike was detected, corresponding to the start of the encryption routine.

The oscilloscope captured the waveforms with each encryption iteration. These waveforms in CSV or binary format stored as a sequence of voltage values sampled with time. An EM trace for each plaintext-key pair is captured. The traces were saved from the oscilloscope in CSV or binary format and converted later to structured numeric arrays.

2.1.3 Structuring the Dataset

Following the successful acquisition of the electromagnetic traces radiated by the Arduino board while encrypting data, the second essential step was structuring the gathered data

into a well-organized dataset amenable to machine learning analysis. Proper structuring of the data is not only crucial for guaranteeing the precision and efficacy of the model training process but also for ensuring traceability and reproducibility in subsequent experiments.

Every electromagnetic trace recorded on the oscilloscope is indicative of a set of voltage values taken over an interval of time, usually illustrating the electromagnetic variability within a single encryption process. The traces themselves, although clearly significant on the oscilloscope display, were stored digitally and associated with unique encryption input variables to create an useful dataset.

To achieve this, the following attributes were included in each data record:

- **Plaintext Input:** This is the message that was encrypted by the Arduino when the EM trace was captured. Adding the plaintext guarantees that the dataset captures the variation in input data and enables researchers to investigate whether specific inputs influence EM leakage patterns.
- **Encryption Key:** Similar to the plaintext, the encryption key employed in each operation is logged to preserve trace-level integrity. In side-channel analysis, key variations can have a strong influence on the emitted EM patterns, and thus capturing this information is crucial.
- **Raw EM Trace:** This is the central part of the dataset. Every trace represents a time series of voltage (amplitude) values sampled by the oscilloscope while encryption was performed. These values were saved as space-separated or comma-separated string of floating-point values indicating the level of electromagnetic radiation over time. In reality, these arrays from hundreds to thousands of values, depending on sampling rate and the duration of the encryption.
- **Derived Statistical Features:** In addition to raw waveform data, basic statistical features were computed for each trace to support feature engineering. These included:

- **Mean:** To capture the central tendency of the signal.
- **Standard Deviation / Variance:** To understand the spread and volatility of the waveform.
- **Signal Range or Amplitude:** To observe how sharp or flat the signal variations are.

After all the data points were gathered and attributes pulled out, the records were saved into a CSV file format. The format was chosen due to its simplicity, portability and compatibility with a broad spectrum of data analysis software like Python (Pandas, NumPy), Excel, and machine learning environments.

The structured CSV dataset was used as input for the preprocessing and model training stages. By keeping it in this form, it became easier to retrain models on new traces, to more easily debug classification mistakes, and potentially to embed this dataset within a larger framework for side-channel vulnerability analysis.

Additionally, this systematic process guarantees that the methodology can be replicated. Other researchers or developers who seek to extend this work can employ the same structure and naming logic, making it easier to collaborate and verify in subsequent research efforts.

2.1.4 Alternative EM Trace Collection Attempt Using Copper Shielded Ethernet Setup

In addition to the standard setup involving the Arduino board and direct probing of the oscilloscope, another method was attempted to explore the potential of capturing electromagnetic emissions in a less direct manner. This experiment arose from the idea of improving trace quality or even experimenting with other methods of EM capture that would be less obtrusive or more sensitive to some types of leakage.

For the test, an Ethernet cable was used as the underlying conductor due to its multi-core structure and shielding. Then a thin copper wire was wrapped around this Ethernet cable

as an impromptu electromagnetic probe. The idea behind this was to create an antenna-like configuration that could pick up EM radiation from the encryption procedure happening inside the Arduino.

The copper coil was then attached to the oscilloscope probe input directly in an attempt for it to detect the electromagnetic interferences in the local environment especially, the leak that is generated by the microcontroller during the course of encryption processes.

Despite theoretical feasibility and effort, this technique was unable to produce visible and usable EM traces. The readings made using the copper-shielded Ethernet environment were extremely distorted, irregular, and often dominated by signal reflections or environmental noise. No pattern was observed that was identifiable with the encryption process or timing, making the data unsuitable for machine learning procedures.

Several factors may have contributed to the failure of this method:

- **Improper grounding or shielding**, leading to excessive noise pickup.
- **Lack of directional sensitivity** since the improvised coil may not have been oriented correctly relative to the source of EM emissions.
- **Low signal strength**, as EM leakage from the Arduino might not have been strong enough to be picked up through this indirect method.
- **Absence of filtering circuitry**, which would typically be present in commercial EM probes to eliminate background interference.

Although this method was not finalized with outcomes, the experiment was an instructional learning experience about the sensitivity and practical limitations of EM trace collection. It indicated the necessity for precise probe placement, signal clarity, and hardware calibration while doing side-channel work.

This approach was eventually abandoned in favor of the simpler oscilloscope probing method, which produced cleaner and more reliable EM traces suitable for analysis and machine learning classification.

2.2 Preprocessing and Feature Extraction

After the raw electromagnetic (EM) traces were gathered through the Arduino-based encryption environment and recorded through the oscilloscope, the subsequent important step in the methodology was preprocessing and feature extraction. This step is important in ensuring that the machine learning model is provided with clean, meaningful, and consistent data for training and testing.

2.2.1 Importance of Preprocessing in Side-Channel Analysis

For the purposes of side-channel analysis, particularly the ones that depend on electromagnetic (EM) emanations, both quality and accuracy of the resultant signal data significantly dictate the potency of any weakness discovery system. Cryptographic procedures during which the EM traces have been obtained usually encounter a diverse host of disturbances originating from within or outside of their environment. These are power supply variations, ambient EM interference caused by other electronic devices in the vicinity, erratic grounding, and electrical noise introduced by the EM signals themselves due to the components of the circuit. Consequently, the raw EM signals are typically noisy and distorted and usually contain irrelevant variations with no useful information describing the encryption process.

Utilizing such raw, unprocessed traces for machine learning can greatly impair the training process. The model can fail to recognize patterns with high variability among traces, or worse, might learn from noise instead of actual side-channel leakage, resulting in bad generalization and misclassification. Preprocessing thus becomes a critical step in the pipeline to improve the signal-to-noise ratio and filter out the real leakage patterns correlated with encryption weaknesses.

The overall aims of preprocessing are to remove noise and irrelevant oscillations, scale the values to ensure consistency across all traces gathered, and normalize the input data structure for subsequent statistical analysis and model training. Reduction in noise makes sure that whatever patterns the model learns are indicative of the encryption process and not of extraneous disturbances. Normalization aids in ensuring consistency across samples

obtained from various sessions or environments, particularly when there are minor discrepancies in voltage ranges or sampling rates. Secondly, preprocessing readies the trace data for effective and reliable feature extraction, a stage that has a direct impact on the performance of the vulnerability classifier.

Finally, without a sound preprocessing phase, even the most advanced machine learning algorithms may not be able to generate useful results. This stage makes sure that the input given to the model is clean as well as rich in information, which makes the model more capable of identifying between vulnerable and non-vulnerable cryptographic implementations with high accuracy.

2.2.2 Cleaning and Denoising EM Traces

The first important stage of the preprocessing pipeline was cleaning and noise removal from the raw electromagnetic (EM) traces obtained through the encryption processes. Since the traces were recorded in the field environment with off-the-shelf hardware, they inevitably had noise and corruption of signals of all forms. These unwanted signals can be due to ambient electromagnetic noise, crosstalk among electronic components, or even minute instability in Arduino voltage regulation. To ensure that the traces could be used effectively to train machine learning, different noise reduction techniques were tried and heavily tested. Each method had some benefits and certain challenges associated with it that had to be considered in detail before the preprocessing pipeline was agreed upon.

Moving Average Smoothing: Moving average smoothing was one of the first denoising methods used. This method takes each data point in the trace and replaces it with the mean of its neighbors over a defined window size. The moving average method is especially good at removing high-frequency noise small, fast changes that obscure the underlying signal trend. In this study, it was noted that the use of moving average smoothing aided in making the overall shape of the EM trace clearer, and hence easier to interpret and analyze.

But while it was fine for fairly clean signals, this approach did have a small downside. The smoothing action created a trace lag and often distorted the start and end of steep transitions especially where the encryption processes generated abrupt voltage spikes.

These lags, as small as they were, might cause the misregistration of critical features in multiple traces. Thus, although the moving average filter was helpful, it needed to be carefully adjusted for window size so that no major changes in the time-domain features of the signal would result.

Median Filtering: In order to deal with some of the weaknesses of the moving average filter, a median filtering technique was also attempted. As opposed to averaging, which smoothes out abrupt changes in the signal, the median filter replaces every point on the trace with the median value of the surrounding points. This method is especially effective in dealing with signal outliers extreme values that occur because of sporadic spikes or blips in the trace resulting from transient hardware noise.

Median filtering did maintain the abrupt edges and transitions in the signal, which were usually associated with precise steps in the encryption algorithm, like S-box lookups or mixing operations in the key. The latter operations may create abrupt changes in EM activity, and keeping such transitions intact was important to maintain the side-channel importance of the trace. Median filtering, however, also needed meticulous exploration of the window length. A small window would preserve much of the noise, whereas a large window could smooth out essential fluctuations. Getting the balance right was critical to preserving trace fidelity while eliminating troublesome outliers.

Low-Pass Filtering: Low-pass filtering was also a method assessed, which lets low-frequency components of a signal pass through and dampens the higher frequencies. It is used frequently in signal processing to filter out fast, jittery oscillations that are usually regarded as noise. A digital low-pass filter applied to the EM traces reduced the effects of very quick voltage fluctuations, which usually resulted from power supply fluctuations or high-frequency EMI from other nearby electronic equipment.

Though the low-pass filter succeeded in quieting the unstable behavior in most traces, it also had an important trade-off. Some of the high-frequency, fine-grained patterns in the EM emissions especially those corresponding to intermediate state changes within the encryption algorithm were suppressed inadvertently. As these slight changes might

represent vulnerabilities, their loss during filtering risked model accuracy. Hence, this approach was utilized only after examining the frequency spectrum of the trace so that significant cryptographic operations were not being concealed by the filtering operation.

One of the biggest challenges during this stage of the investigation was how to balance the trade-off between noise removal and signal integrity. Most filtering methods, by being good at trace cleaning, risked over-smoothing the data and removing fine but significant variations. In early trials, too much aggressive denoising caused loss of trace features later found to be essential for classifying vulnerability. This rendered the machine learning model less susceptible to key leakage patterns.

In addition, not all the traces reacted to filtering equally. Some EM traces were inherently more stable based on environmental conditions, whereas others were extremely erratic. Consequently, there was no single solution that fit all. Dynamic adjustment of filtering parameters or using various combinations of filters depending on the trace nature was necessary. This trial-and-error, iterative process constituted a major portion of the preprocessing stage and took a lot of effort to optimize.

2.2.3 Normalization and Scaling

After the EM traces were cleaned and denoised, normalization and scaling were the next necessary step in preprocessing. Raw EM trace data recorded in various sessions or in slightly different environmental conditions may have different voltage ranges owing to oscilloscope calibration, circuit board characteristics, or even variations in USB power. These variations, while occasionally minute to the visual eye, potentially could have a great impact on the consistency of input data to machine learning models, which are very sensitive to input scales. As such, normalization was applied so that all traces had a common scale and statistical behavior so that the model would be concerned with meaningful patterns and not absolute signal values.

Min-Max Normalization- The major method employed in this study was Min-Max Normalization, where every trace was scaled to a uniform range particularly between 0

and 1. This was achieved by recognizing the smallest and highest voltage readings in every EM trace.

This method guaranteed that all trace values were normalized into a common scale, allowing for consistent comparison among different encryption operations and sessions. Min-Max normalization also benefits from the added feature of preserving signal shape and relative spacing, which is important in side-channel analysis since the time-domain structure of the trace holds vulnerability-related information.

In addition, models of neural networks like fully connected layer or convolutional architectures tend to generalize better and converge more quickly if input values fall within a finite range. Using this method, the model training became more stable and effective, with fewer opportunities for numerical instability due to excessively large or randomly scaled input features.

Z-score Standardization- Apart from Min-Max scaling, Z-score Standardization was also considered as an alternative, especially during the experimental process. This method involves converting the trace values in terms of their mean and standard deviation.

This procedure brings the data around zero and rescales it according to its statistical range. Z-score normalization is particularly useful when there is a widely varying data distribution or when machine learning algorithms are sensitive to feature distributions, as with Support Vector Machines (SVM) or some tree-based algorithms.

But in practical application, this technique had a couple of drawbacks for the case of EM side-channel analysis. With traces that were already fairly clean and showed uniform ranges, the Z-score transformation occasionally boosted small outliers. These could either be benign or noise-induced outliers, which were accorded greater significance compared to the remainder of the trace, and the model could then misinterpret them. In some instances, it also imposed negative values and non-uniform scaling, which had a minimal impact on the visual interpretability of the EM waveform.

The decision between Min-Max and Z-score normalization was not a technical one it was iterative testing, comparison, and weighing the requirements of trace integrity against model behavior. Although Min-Max was eventually chosen as the method of choice because of its simplicity and trace-preserving nature, it did necessitate ensuring that no session created extreme outliers that would skew the range. Also, it was essential to apply the same normalization approach uniformly to both the training and test datasets to eliminate mismatches that might cause biased prediction results.

In short, normalization was essential in data preparation for machine learning. It made the dataset consistent and allowed the classifier to train on patterns instead of noise or scale variation caused by hardware. The comparison of both normalization techniques gave strength to the preprocessing step, eventually ending up with a cleaner and better input for feature extraction and model training.

2.2.4 Feature Engineering

After the EM trace cleaning and normalization, feature engineering was the following vital step of the preprocessing process. Feature engineering is a methodology for converting raw data into form inputs that make the model a better predictor. For the sake of this study, feature engineering is recognizing essential EM trace characteristics capable of distinguishing between weak and secure encryption operations.

Since side-channel leakages particularly electromagnetic emissions are likely to appear as faint and transient anomalies in the trace, the appropriate set of features can greatly improve model performance. Rather than passing the raw, high-dimensional waveform directly into the model, chosen features were extracted and saved in a compact form for use in training and testing. These features are categorized into three main groups: statistical features, time-domain features, and frequency-domain features.

2.2.4.1 Statistical Features

Statistical characteristics were among the strongest and most credible indicators in the study. However, the characteristics were developed from the amplitude values of a single EM trace and assisted in quantifying common patterns in signal behavior.

- **Mean Value:** This is the mean voltage level across the trace as a whole. In preliminary exploratory analysis, traces for encryption operations which had been specifically weakened or unprotected tended to have subtle variations in their mean signal levels. This rendered the mean an important feature for classification and even for initial threshold-based tagging in the construction of the dataset.
- **Standard Deviation:** A measurement of the trace's dispersion or variability. A large standard deviation may suggest a fast rate of changes, which may translate to hard or chaotic hardware activity that is likely indicative of information leakage.
- **Skewness and Kurtosis:** These are second-order statistical measures. Skewness measures the asymmetry of the trace distribution, and kurtosis measures the "peakedness" of the signal. Highly skewed or sharply peaked traces commonly indicated irregular or abnormal behavior during encryption, such as bursts of computation. These characteristics added a new dimension to the model's knowledge of the trace characteristics.
- **Maximum and Minimum Values:** These capture the outermost points of the trace's amplitude. In some encryption operations, hardware operations such as S-box lookups or key mixing can result in recognizable high-voltage spikes or dips. Recording these extremes were necessary in order to detect deterministic leakages corresponding to internal hardware operations.

2.2.4.2 Time-Domain Features

In addition to basic statistics, a number of time-domain features were extracted to provide insight into the structure and dynamics of each trace.

- **Peak-to-Peak Distance:** This characteristic calculates the time difference between significant peaks in the trace. Because encryption algorithms usually involve repeated rounds of computation, the interval between these bursts of energy might indicate regularity or irregularity of internal processes. Deviations from this

pattern occasionally aligned with the implementation of optimized vs. non-optimized encryption code, which influences vulnerability.

- **Signal Energy:** Calculated as the sum of the amplitude values squared, the energy of the trace indicates the extent to which "work" the device was doing when encrypting. Overall, the operations that were more energy intense in bursts were marked as being potentially more leaking information because they had greater switching activity on the chip.

These features in the time domain proved to be so useful since they maintained temporal dependency within the trace so that the model could notice not only that there was such fluctuation but when and with what frequency it occurred.

2.2.4.3 Frequency-Domain Features (Explored but Not Used)

As a part of exploratory data analysis, frequency-domain features were also explored with the Fast Fourier Transform (FFT). The notion was to convert every EM trace from the time domain to the frequency domain to examine its spectral components. The hypothesis was that insecure encryption implementations could produce repeating frequency patterns due to periodic computation cycles that could be used as fingerprint-like indicators of leakage.

However, several challenges emerged during this process:

- The resolution of the oscilloscope used for data collection was relatively low, limiting the clarity and precision of frequency components.
- Environmental and electrical noise often overlapped with useful frequency bands, leading to noisy FFT plots.
- Feature vectors derived from the FFT were high-dimensional and sparse, which added computational overhead without noticeable improvement in model performance.

Due to these constraints, frequency-domain features were excluded from the final model training pipeline. Nevertheless, this exploration highlighted the potential of spectral analysis in future work where higher-grade equipment and better signal isolation are available.

One of the key challenges faced during feature engineering was finding the right balance between information richness and computational efficiency. Including too many features especially redundant or noisy ones risked overfitting and increased training time. Conversely, too few features could under-represent the complex nature of side-channel leakages. Several iterations were conducted to assess the usefulness of different feature combinations through trial-and-error, correlation studies, and early testing using cross-validation.

Another challenge was ensuring feature consistency across different EM trace sessions. Since even minor variations in hardware behavior can alter trace properties, care was taken to apply the same feature extraction procedures uniformly across all data to avoid introducing bias or inconsistencies into the model.

In summary, the feature engineering process transformed raw, high-dimensional EM signals into structured, lower-dimensional representations that encapsulated the most informative aspects of the traces. The carefully selected features primarily statistical and time-domain enabled the machine learning model to accurately differentiate between vulnerable and secure encryption behaviors, ultimately forming the foundation of the system's detection capability.

2.2.5 Labeling the Dataset

Labeling the dataset was a crucial part of the preprocessing pipeline, as the overall accuracy and trustworthiness of the whole machine learning model heavily rely on the quality of training labels. As the aim of this research was to classify EM traces as either vulnerable or non-vulnerable, a procedure had to be adopted to label every trace accordingly. However, because there was no ground truth information available in the form

of known leakage models or previous successful attack attempts an alternative labeling technique needed to be constructed.

After a rigorous examination of many EM traces, it was seen that some statistical characteristics, mostly the mean level of voltage for a trace, always indicated the nature of the cryptographic operation. Traces resulting from insecure implementations (e.g., without correct side-channel countermeasures) would often indicate signal distortions or energy pulses that moved the overall signal average beyond a common range.

On the basis of these findings, the mean value of each EM trace was chosen as the main measure for labeling.

2.2.5.1 Labeling Methodology

To provide adequate labels to all electromagnetic (EM) traces, the mean of each trace was computed while doing feature engineering. The mean was then used as the baseline classifier. A threshold approach was adopted where traces with mean values less than 0.45 or greater than 0.54 were labeled vulnerable, and traces with mean values between 0.45 and 0.54 as non-vulnerable. These threshold values were not random but were determined from an empirical trace distribution derived from traces obtained over multiple encryption sessions. Repeated visual inspection of oscilloscope signal behavior, as well as statistical analysis, revealed that secure implementations of encryption tended to produce stable and centered EM signatures with similar average voltage levels. Traces from potentially insecure implementations, on the other hand, tended to exhibit large deviations from this range, indicating anomalous power or processing activity. This labeling method provided an efficient and practical method of trace data classification without ground-truth attack results and encouraged consistency in the dataset.

2.2.5.2 Why This Approach Was Necessary

In typical side-channel attacks, marking is typically performed based on a foundation of knowledge of secret key bytes or dummy attack situations. Within this hardware-based research environment particularly when time and resource limited it was not possible to conduct actual attacks in an effort to determine if each trace was actually susceptible.

So statistical labeling with mean thresholds was born as a viable and scalable solution. It allowed for automatic labelling of an enormous volume of data and obtaining a reasonable level of confidence in the labelling outcome.

This method was especially suitable for this research because:

- It aligned with observed EM trace behavior during real encryption operations.
- It avoided the need for intrusive or time-consuming side-channel attacks.
- It enabled repeatability across different datasets and environments.

One of the major challenges in this phase was determining an appropriate threshold range for labeling. If the range was too narrow, many potentially non-vulnerable traces would be falsely labeled as vulnerable, leading to a high number of false positives. On the other hand, if the range was too wide, subtle leakages might go undetected, resulting in false negatives.

To overcome this, an iterative thresholding process was adopted. This involved:

- Plotting the distribution of trace means using histogram visualizations in Python.
- Manually inspecting traces with means near potential threshold boundaries using the oscilloscope.
- Repeating this process across several encryption sessions to ensure consistency.

The subjective nature of visual inspection was mitigated by gradually automating the labeling process using a Python script, which will be further discussed in a later section. This script not only calculated the mean for each trace but also assigned labels based on the defined thresholds, ensuring uniform labeling across the dataset.

2.2.5.3 Labeling Implementation

To systematically and consistently label the dataset based on trace characteristics, a Python script was developed to automate the classification of electromagnetic (EM) traces as

either Vulnerable or Non-Vulnerable. This automation helped reduce subjective errors introduced during manual inspection and ensured reproducibility of the labeling process.

The script begins by loading the raw dataset containing plaintext, encryption keys, and EM trace data. Each EM trace, initially stored as a string of space-separated voltage values, is parsed into a numerical array using NumPy for further analysis. Once parsed, the mean of each trace is computed this statistical measure served as the primary feature for labeling vulnerability.

Based on prior empirical analysis and visual inspection of EM signals, two thresholds were defined:

- A lower threshold of 0.45
- An upper threshold of 0.54

Traces with a mean value outside this range (i.e., less than 0.45 or greater than 0.54) were classified as Vulnerable, while those within the threshold range were labeled as non-vulnerable. This binary classification was then added as a new column in the dataset, labeled "Vulnerable".

The use of this automated script ensured a consistent, threshold-based labeling process that was grounded in signal behavior observed during earlier experiments. It also made the dataset more suitable for supervised learning by clearly associating each trace with a definitive label.

2.2.5.4 Challenge Faced

Amongst the biggest hurdles in this step was how one was to pick a good enough range of thresholds to label them as. A too narrow a range would include too many supposedly non-vulnerable traces as being vulnerable, meaning that there will be a plethora of false positives. On the other hand, a too big a range of thresholds would hide the small leakages, thereby there will be false negatives.

To overcome this, an iterative thresholding process was adopted. This involved:

- Plotting the distribution of trace means using histogram visualizations in Python.
- Manually inspecting traces with means near potential threshold boundaries using the oscilloscope.
- Repeating this process across several encryption sessions to ensure consistency.

The inbuilt subjectivity of visual inspection was eased by increasingly automating the labeling process using a Python script, which will be discussed in a later section. The script not only calculated the mean value per trace but also labeled appropriately depending on predetermined thresholds such that labeling across the dataset was uniform.

2.2.5.5 Future Considerations

While the threshold-based labeling method was satisfactory for the scale of this study, there is potential to enhance labeling's accuracy and robustness in future studies. One is to add semi-supervised learning techniques. In this instance, a limited amount of traces with ground truth labels verified by professional analysis or empirical side-channel attack outcomes may guide the classification of a larger, unlabeled collection. This would reduce dependence on manually set thresholds and improve model generalization. Another probable path is that of employing unsupervised cluster algorithms, such as k-means or DBSCAN, to enable automatic identification of natural clusters from the EM trace data. The algorithms would identify unsuspected structure within the set and cluster the traces based on similarity inherent in them, possibly being able to see patterns not seen with thresholding. Furthermore, performing differential electromagnetic analysis (DEMA) in future experiments can confirm the labeling method against actual cryptographic leakage. This would offer a way of checking if traces labeled as "vulnerable" indeed leak exploitable information and ensure that the detection system aligns with realistic side-channel attack capabilities. Such enhancements could lead to an adaptive, scalable, and reliable system for real-world deployment.

Finally, the mean-based labeling approach showed a correct, uniform, and thrifty way of getting a large-volume dataset ready to utilize in training models. While defining the

thresholds was troublesome initially, the approach turned out to separate trace types properly in a way that accommodated hardware behavior perfectly in practice.

2.2.6 Structuring the Preprocessed Dataset

After completing the signal cleaning, normalization, feature extraction, and labeling phases, the final step in the preprocessing workflow involved structuring the dataset in a way that was both systematic and machine learning friendly. This phase was not just about organizing the data it was also about creating a reusable and scalable format that preserved all contextual information associated with each trace. The structured dataset became the foundation for training and validating the machine learning model and enabled further exploration and fine-tuning of the detection system.

The preprocessed data was stored in Comma-Separated Values (CSV) format. CSV was chosen because it offers maximum compatibility with data analysis and machine learning libraries such as Pandas, NumPy, Scikit-learn, and TensorFlow, all of which were used at various stages of experimentation. Additionally, CSV made it easier to manually inspect the dataset during debugging and exploratory data analysis, which was especially useful when verifying the accuracy of the labeling algorithm and statistical calculations.

Each row in the CSV file corresponded to a single encryption event and contained the following columns:

- **Plaintext:** The original input plaintext (usually hexadecimal characters) that was supplied to the Arduino during encryption. This field helped maintain traceability, especially when repeating experiments or validating how changes in plaintext impacted the shape of the EM trace.
- **Encryption Key:** The 16-character hexadecimal key used in the encryption operation. Logging this was crucial for verifying whether specific key patterns had a measurable impact on EM leakage.
- **Raw EM Trace:** This column contained the voltage readings captured directly from the oscilloscope, typically in the form of a long, comma-separated array of

float values. These raw signals were preserved in the dataset to allow re-analysis or reprocessing in the future using alternative techniques. Storing raw traces also provided a fallback in case the preprocessing pipeline needed to be revised.

- **Parsed EM Trace:** This field held the cleaned and preprocessed version of the raw trace. Depending on the preprocessing configuration (e.g., low-pass filtering, normalization), the length of this array might vary slightly. These traces were used directly in feature engineering and model training, so their presence in the dataset helped reduce redundant preprocessing in downstream stages.
- **Mean:** The average voltage value of the parsed EM trace, which played a dual role as a statistical feature and as a key input in the threshold-based labeling mechanism. This value was calculated automatically by a Python script during preprocessing.
- **Vulnerability Label:** Based on the computed mean, a binary label was assigned to each trace: either Vulnerable or Non-Vulnerable. These labels were essential for supervised machine learning and were derived according to empirically established thresholds.

2.2.6.1 Setup Considerations and Implementation Challenges

Constructing this formal dataset was not without issues. One of the very initial issues encountered was trace alignment. Since traces collected from the oscilloscope had tiny timing drifts depending on the processing latency of the Arduino board and triggering sensitivity, traces needed to be normalized in terms of lengths. This was either achieved by zero-padding shorter traces or truncating longer traces to a set length so that all parsed traces shared the same dimensions for model input compatibility.

Another important problem was script automization. Initially, the entire preprocessing workflow trace parsing, cleanup, feature generation, and marking was performed manually in a Jupyter Notebook. But once the dataset became larger, that became

impossible. Therefore, a modular Python script was built to batch all trace files within a special directory and export the structured CSV file automatically. Not only did this script accelerate the process but it also ensured that every transformation was homogeneous across the whole dataset, keeping human error under control.

Special configuration had to be applied even to the files organization of the dataset. The EM traces were initially stored in individual.txt files that had been exported out of the oscilloscope's user interface. These trace files were imported into NumPy arrays, and more code was added to pair each trace with its associated key and plaintext values. For proper indexing and tracing, a naming convention was adopted for trace files (e.g., trace_keyX_plainY_indexZ.txt), which helped the script join metadata during CSV generation.

One of the challenges during structuring was storing long waveform arrays in a single CSV cell. Since every EM trace can have thousands of data points, saving them in an optimal way without breaking the CSV format took some intelligent formatting. Arrays were transformed into a single string of float values comma or semicolon-separated and surrounded by quotes to be read correctly. Even though this did add some weight to the file, it ensured waveform integrity and compatibility with Python CSV reader.

Finally, intensive sanity checks were added to the preprocessing script to detect missing or broken trace files, verify data consistency (e.g., verifying that mean values aligned with the correct trace), and verify that each key-plaintext pair had an associated EM trace and label. Not only did this render the final product suitable for direct model training but also made the resulting dataset resilient for subsequent experiment and audit tasks.

By the time this stage was over, a systematically organized and traceable dataset was created a crucial milestone that laid the groundwork for the transition towards raw experimental data to smart side-channel vulnerability identification. The structuring of the data made it reusable, build upon, and verifiable, laying the ground for model improvement and reproducibility to be attained at the next stages of the project.

2.3 Machine Learning Model Training

After completing the preprocessing pipeline which involved cleaning, normalization, feature extraction, and labeling the next phase focused on developing a machine learning model capable of automatically classifying electromagnetic (EM) traces as either Vulnerable or Non-Vulnerable. The purpose of this classification was to detect whether the input trace, obtained during the encryption process, exhibited characteristics that made it susceptible to side-channel attacks.

The overarching goal was not only to achieve high accuracy but also to ensure the model was computationally lightweight and suitable for deployment on embedded systems like the Raspberry Pi. This requirement introduced a key challenge: striking the right balance between model complexity and hardware efficiency. While more complex models (like deep learning networks) may offer marginal accuracy improvements, they often require more resources than what embedded platforms can support. Therefore, classical machine learning algorithms were prioritized during this phase.

The model training process began with the preparation of the dataset. From the earlier steps, each data instance consisted of a parsed EM trace which is a numerical sequence representing fluctuations in electromagnetic activity and a corresponding vulnerability label which is "Vulnerable or Non-Vulnerable", determined using threshold-based statistical labeling.

These two elements formed the input features (X) and the target variable (y) for the model, respectively. One of the main challenges during this step was the dimensionality of the input data each parsed EM trace contained dozens (or even hundreds) of float values, and feeding the entire trace into the model without proper dimensionality reduction could lead to:

- Overfitting on training data
- Increased training time
- Poor generalization on new, unseen traces

To overcome this, some preliminary feature reduction strategies were explored (such as slicing or focusing on key trace segments), although the final model still relied on using the core EM trace values as the primary features.

Before model training, the dataset was split into training and testing subsets using an 80-20 ratio, a common practice to evaluate how well the model generalizes to unseen data. This means 80% of the dataset was used to train the model, while the remaining 20% served as a validation set to test its performance. A challenge encountered here was ensuring that the label distribution remained balanced across both subsets. In early attempts, random splits occasionally led to imbalanced distributions where one class (e.g., "Vulnerable") dominated, which skewed performance metrics. This was later resolved by using stratified splitting to preserve class ratios.

Overall, this stage laid the foundation for building a deployable and efficient classification model by preparing the input-output structure, managing data dimensionality, and ensuring proper data splitting for unbiased evaluation.

2.3.1 Model Selection

Once the dataset was structured and split appropriately, the next step was selecting a suitable machine learning model for the classification task. Given the resource constraints of embedded systems like the Raspberry Pi, it was crucial to avoid computationally intensive deep learning models and instead focus on classical machine learning algorithms that are known for their efficiency, interpretability, and ease of deployment.

Several well-established models were considered, each offering a different approach to classification. The following models were shortlisted and tested:

2.3.1.1 Random Forest

Random Forest is an ensemble learning algorithm that constructs multiple decision trees during training and combines their outputs to make a final prediction. By aggregating the predictions of several trees, it effectively reduces the risk of overfitting a common issue with individual decision trees while maintaining high classification accuracy. One of the

core strengths of Random Forest lies in its ability to handle high-dimensional and noisy datasets, which makes it well-suited for real-world applications like EM trace classification, where the data can be inherently variable and noisy. Moreover, the model provides insights into feature importance, helping to interpret which statistical aspects of the EM signals (such as mean, variance, or peak amplitude) contribute most to identifying vulnerabilities. However, the use of multiple decision trees comes at a cost: the model demands more memory and has a longer training time compared to simpler algorithms. Although it is more computationally intensive than Logistic Regression, careful tuning of the number of trees and tree depth helped to strike a balance between inference speed and accuracy, making Random Forest a viable candidate for embedded deployment on Raspberry Pi.

2.3.1.2 Support Vector Machine (SVM)

Support Vector Machine is a powerful binary classifier that attempts to identify the optimal hyperplane that separates two classes with the maximum possible margin. This characteristic makes SVM particularly effective for high-dimensional datasets, where it can often find clear separation boundaries even when features are numerous and overlapping. The algorithm performs well when the classes are separable and the data distribution is balanced, which was observed to some extent in the EM trace dataset after preprocessing. However, the computational demands of SVM pose a significant challenge especially during inference since it relies heavily on calculating distances from support vectors. This leads to slower predictions, particularly when a large number of support vectors are involved, which is not ideal for real-time classification tasks on low-powered hardware such as Raspberry Pi. Additionally, the lack of native probabilistic outputs and the need for extra computation to enable confidence scoring limited its practicality for deployment in this research. While SVM yielded relatively strong performance during offline evaluation, these constraints made it less suitable for the final embedded application.

2.3.1.3 Logistic Regression

Logistic Regression is a widely used linear model for binary classification tasks, offering simplicity, interpretability, and low computational overhead. It models the relationship between input features and the target label using a logistic (sigmoid) function, which makes it particularly fast to train and deploy. Due to its low memory footprint and straightforward implementation, Logistic Regression was an attractive choice for early prototyping and benchmarking. In addition, the linear nature of the model makes it easy to debug and understand, providing clear insights into how each feature affects the final classification decision. However, this same linearity becomes a limitation when the underlying relationships in the data are non-linear as was the case with the EM traces used in this research. Although Logistic Regression delivered acceptable accuracy, it failed to identify certain subtle variations within the vulnerable class, resulting in lower recall. This trade-off between simplicity and predictive power rendered Logistic Regression an excellent baseline but ultimately insufficient for achieving the desired accuracy in vulnerability detection.

2.3.1.4 K-Nearest Neighbors (KNN)

K-Nearest Neighbors is an intuitive and non-parametric algorithm that classifies data based on the majority label of the 'k' most similar instances from the training dataset. It requires no explicit training phase, as the model essentially memorizes the training data and performs computations only during inference. This characteristic makes KNN easy to implement and highly flexible for problems where local patterns dominate global trends. However, this advantage becomes a drawback in resource-limited environments, as every prediction requires calculating the distance from the input to every training sample. In the case of EM trace analysis, where the dataset could grow significantly, KNN's inference time became impractically long for real-time embedded applications. Furthermore, its performance degraded as dataset size increased, and its storage demands were unsuitable for Raspberry Pi-based systems. While KNN did yield reasonable accuracy when tuned properly, the computational burden and scalability issues ultimately excluded it from being a viable candidate for final deployment in this research.

2.3.2 Final Model Selection and Evaluation

To determine the most appropriate model for deployment in the embedded classification system, all four shortlisted machine learning algorithms Random Forest, Support Vector Machine (SVM), Logistic Regression, and K-Nearest Neighbors (KNN) were evaluated using a combination of standard performance metrics. To fairly assess the performance of the machine learning models used in this research, several standard evaluation metrics were employed. These metrics provided a comprehensive understanding of how well each model performed, particularly in the context of classifying electromagnetic (EM) traces as either Vulnerable or Non-Vulnerable.

Accuracy was the primary measure of overall performance. It calculated the proportion of correctly predicted instances out of the total predictions made by the model. While accuracy gives a general sense of correctness, it can sometimes be misleading in cases where the dataset is imbalanced especially when one class dominates. Hence, accuracy was used alongside more focused metrics to gain a nuanced view of performance.

Precision evaluated the quality of positive predictions. Specifically, it measured the proportion of traces that the model labeled as "Vulnerable" that were truly vulnerable. A high precision value indicated that the model made few false positive errors, which is critical in side-channel attack detection scenarios where incorrectly flagging safe traces as vulnerable can lead to unnecessary concerns or wasted mitigation efforts.

Recall focused on the model's ability to capture actual vulnerable cases. It represented the proportion of truly vulnerable traces that the model successfully identified. High recall was important in this research context, as failing to detect a vulnerable trace (a false negative) could result in a missed threat, potentially compromising the system's security.

F1-Score served as the harmonic mean of precision and recall. It provided a single, balanced metric that considered both false positives and false negatives, making it particularly useful when the dataset exhibited slight class imbalance. A strong F1-score indicated that the model was not only accurate in its predictions but also reliable in both detecting vulnerabilities and avoiding incorrect alerts.

Cumulatively, these four measures constituted a balanced evaluation framework. They allowed for in-depth comparison of model performance, ultimately deciding on the best algorithm to implement on the embedded system. Prioritizing detection accuracy and prediction efficacy, the study ensured that the ultimate system was security relevant and functionally deployable.

All models were evaluated with an 80-20 train-test split for equitable testing and emulating generalization in real-world scenarios. Initial experiments showed some performance vs. resource usage trade-offs. For example, though computationally lighter models such as Logistic Regression and SVM performed better, they performed poorly in detecting complex patterns in the EM traces. KNN, despite the simplicity in concept, was scalable and slow at prediction due to its instance-based approach, which was a performance bottleneck during inference.

Following a few iterations of training and validation, Random Forest emerged as the most effective and practical model. It consistently generated the top or second-highest values across all the most significant performance metrics, particularly in F1-Score, which represented its balanced ability to detect both vulnerable and non-vulnerable traces without leaning toward either class. Its ensemble nature provided it with the capability of generalizing more on unseen data and avoiding overfitting a common issue in noisy real-world EM datasets.

Whereas Random Forest was slightly more memory-consuming and computationally heavy in training time compared to more rudimentary models, such drawbacks were acceptable in light of the Raspberry Pi 4 hardware specification. Through fine-tuning the number of decision trees and limiting the maximum tree depth, the model size and execution were tailored to conform within real-time inference needs. Further, its support for feature importance analysis provided insightful information regarding which of the EM trace statistical features were most indicative and provided a second level of interpretability to the classification procedure.

The final Random Forest choice was thus not solely based on unprocessed performance but also with practical deployment feasibility in mind. It was the optimal compromise between accuracy, speed, and resource usage and was thus an ideal candidate for inclusion in the Raspberry Pi-based hardware prototype. This ensured that the system could classify incoming EM traces in real-time with a high level of reliability, fitting the research aim of creating a portable and efficient side-channel vulnerability detection device.

2.4 Implementing on Hardware (Raspberry Pi Deployment)

The choice of the appropriate hardware platform was also a crucial step towards bridging the side-channel vulnerability detection system from software simulation to a real, portable working device. The most essential requirements were straightforward: the chosen hardware must provide light-weight machine learning inference support, signal processing workflow support, GPIO interfacing support for the display components, and be affordable enough to be viable within low-scale security auditing or educational purposes. The solution also needed to be portable and energy-efficient in order to make the system deployable in limited environments.

To meet these goals, a series of candidate platforms were evaluated for their feasibility as follows.

2.4.1 Laptop-Based Setup

In the early stages of system development, the possibility of deploying the trained machine learning model on a standard laptop was explored. Laptops offer a reliable and resource-rich computing environment, featuring powerful CPUs, ample RAM, and native support for full-fledged operating systems like Windows or Linux. These attributes made laptops ideal for initial prototyping. In fact, during the preliminary phases of experimentation and model testing, Python scripts and Jupyter Notebooks were extensively used on a laptop to analyze EM trace data, perform feature extraction, visualize patterns, and fine-tune the machine learning model. The flexibility and debugging convenience offered by a laptop environment significantly accelerated the model development process.

However, as the research advanced toward developing a standalone, portable solution for real-time side-channel vulnerability detection, the drawbacks of relying on a laptop became evident. The most significant limitation was portability. Laptops, although compact compared to desktop systems, are still not ideal for embedded applications or field deployment. They are not designed to be carried around as dedicated embedded devices and require continuous power, making them unsuitable for battery-operated or compact installations.

Another major drawback was the lack of GPIO (General Purpose Input/Output) pin access. In this research, displaying the vulnerability detection result on a physical LCD screen was a key requirement. Standard laptops do not offer native GPIO interfaces, which are essential for directly controlling or communicating with external hardware components like displays, sensors, or actuators. Any such hardware integration would require additional USB-to-GPIO converters or external microcontrollers, further complicating the setup.

Additionally, power consumption posed another concern. Laptops consume significantly more power than embedded platforms such as microcontrollers or single-board computers. Running a machine learning inference system continuously on a laptop would not only increase energy demands but also limit the feasibility of long-term or unattended deployments. The overall size, power requirements, and lack of peripheral control options made the laptop-based approach impractical for real-time, standalone side-channel analysis.

While laptops served as an excellent development platform during the model training and debugging phase, they were ultimately ruled out for the final hardware implementation. The need for a more compact, energy-efficient, and hardware-integrated platform led to the exploration of more suitable embedded systems, which are discussed in the subsequent sections.

2.4.2 ESP32 and Other Microcontroller Boards

Following the limitations identified in the laptop-based setup, the research shifted focus to microcontroller platforms, which are commonly used in embedded systems and IoT applications. Among the many available microcontrollers, the ESP32 stood out as a compelling candidate due to its affordability, compact size, and built-in wireless capabilities. The ESP32 is powered by a dual-core Tensilica LX6 processor and comes with integrated Wi-Fi and Bluetooth, making it a widely adopted choice for small-scale, low-power applications.

From a design perspective, the ESP32 offered several significant advantages. Firstly, its low cost made it an ideal option for research constrained by a limited hardware budget. Boards based on ESP32 are priced affordably and are widely available, reducing the barrier to experimentation. Secondly, the portability of the ESP32 was a major appeal. With a small footprint and extremely low power consumption, it is well-suited for battery-operated deployments, aligning well with the vision of building a field-ready side-channel detection tool. Additionally, ESP32 offers a wide range of GPIO (General Purpose Input/Output) pins, enabling direct interfacing with peripherals such as LCD displays, buttons, sensors, and LEDs, which could support a fully interactive and hardware-integrated implementation.

Despite these advantages, the ESP32 presented several critical limitations that made it unsuitable for the machine learning component of this research. One of the most significant constraints was limited memory. ESP32 typically has around 520KB of SRAM and 4MB of Flash memory, which imposes severe restrictions when attempting to deploy even lightweight machine learning models. Running standard models trained using popular frameworks like scikit-learn or TensorFlow would require aggressive model compression techniques such as pruning, quantization, or conversion to TensorFlow Lite Micro. Even with such optimizations, the memory ceiling of ESP32 made it difficult to reliably load models and inference data, especially when handling multi-feature inputs derived from EM traces.

Another key issue was the lack of full Python support. While the ESP32 can run MicroPython, this subset of Python lacks support for widely used data science and machine learning libraries such as NumPy, Pandas, or scikit-learn. As a result, model inference and feature-based logic would require either rewriting models in C++ (via frameworks like TensorFlow Lite Micro) or severely limiting the complexity of computations. This would compromise the research goals and increase development complexity.

In addition to these issues, the ESP32 also lacked robust support for file handling and CSV parsing, which are crucial for managing EM trace data. Since this research involved loading, parsing, and processing datasets containing trace features and labels, the absence of native or efficient file I/O made it difficult to work with stored data in a structured format. This limitation further complicated attempts to port the model and associated logic to the ESP32.

In summary, although the ESP32 showed promise in terms of cost, size, and GPIO capability, its inability to support full machine learning workflows, restricted memory, and limited software support rendered it unsuitable for the current stage of this research. However, the ESP32 remains an intriguing option for future versions of this system, particularly if the model is optimized further or if a lightweight heuristic approach replaces the ML-based classifier. Its efficiency and power-saving capabilities could prove useful in developing ultra-low-power or wearables-based detection tools in the future.

2.4.3 NVIDIA Jetson Nano

The NVIDIA Jetson Nano was also evaluated as a potential hardware platform due to its reputation as a powerful single-board computer specifically designed for AI and edge computing tasks. The Jetson Nano is equipped with a quad-core ARM Cortex-A57 CPU and a 128-core Maxwell GPU, providing hardware-level support for complex neural networks and real-time data processing. This architecture makes it particularly appealing for applications involving computer vision, object detection, robotics, and deep learning inference. It supports full-featured frameworks like TensorFlow, PyTorch, Keras, and

OpenCV, all of which can run natively with GPU acceleration. In addition, NVIDIA's JetPack SDK includes a complete suite of developer tools, libraries (e.g., cuDNN, CUDA), and pre-configured environments optimized for deploying AI models on the device.

Given these capabilities, the Jetson Nano was initially considered as a strong candidate for deploying the EM trace classification model. Its ability to handle GPU-accelerated operations promised faster inference times and scalability if future versions of the system required deeper or more complex neural networks. Moreover, the development experience on the Jetson Nano is well-documented, and the availability of community support, tutorials, and accessories made it a practical and developer-friendly option.

However, upon deeper analysis, several limitations emerged that ultimately disqualified the Jetson Nano for this particular research. The first and most significant issue was that the Jetson Nano's capabilities far exceeded the requirements of the task at hand. The model used in this research was relatively lightweight, based on statistical features of EM traces, and did not involve computationally expensive deep learning architectures. The use of GPU acceleration, while beneficial in high-load AI tasks, would not provide a substantial performance benefit for this model. Thus, deploying such a simple classifier on a high-performance board like the Jetson Nano was considered overkill.

Secondly, and more critically, the Jetson Nano introduced concerns around cost and power efficiency. Compared to the Raspberry Pi, the Jetson Nano is significantly more expensive, both in terms of the board itself and the additional components it requires for stable operation. It typically demands a dedicated 5V 4A power supply, and this added hardware requirement contributes to both setup complexity and increased cost. The board's power consumption is also higher than more compact alternatives, making it less suitable for a portable, battery-powered device intended for practical field deployment. In scenarios where energy efficiency and affordability are priorities as in this research the Jetson Nano proved to be less aligned with the project's goals.

In conclusion, while the Jetson Nano is a powerful and capable platform for edge AI deployments, it was deemed unsuitable for this research due to its unnecessary performance capacity, higher cost, and greater power requirements. The classification task's simplicity did not justify the investment in GPU-based computation, and a more balanced solution like the Raspberry Pi 4 was ultimately selected for its adequate performance and cost-effectiveness.

2.4.4 Final Selection – Raspberry Pi 4

After an in-depth evaluation of a number of different hardware platforms for executing the trained machine learning system, the Raspberry Pi 4 ultimately became the optimal available solution for this study due to a consideration of numerous factors such as processing power, energy efficiency, interfacing hardware, cost, and software availability. Compared to heavier laptop-based systems and RAM-limited microcontrollers, the Raspberry Pi 4 offered the ideal middle ground with a strong, light-weight computing platform suitable for both real-time inference and hardware control.

The Raspberry Pi 4 features a 64-bit quad-core ARM Cortex-A72 processor at 1.5 GHz, and one used in this research had 4GB of LPDDR4 RAM. This setup was more than sufficient to process the trained classification model, statistical feature-based and not requiring deep learning libraries or GPU acceleration. The board performed the computational tasks extremely well, processing EM trace data and producing predictions with no observable performance bottlenecks.

One of the standout features of the Raspberry Pi is the presence of a full-fledged Linux operating system (Raspberry Pi OS). This environment facilitated easy operation of Python-language code and operation of popular machine learning libraries such as scikit-learn, NumPy, pandas, and joblib (utilized in loading the trained model). In comparison to tiny microcontrollers such as ESP32, which oppose stripped-down interpreters such as MicroPython, the Raspberry Pi accommodated an identical development platform as a common PC, largely simplifying deployment and reducing debug time.

One of the major technical needs of the research was the ability to interface with external hardware, in this case an LCD screen used to display the model's predictions of vulnerability. The GPIO pins on the Raspberry Pi made it an ideal choice for this kind of hardware interfacing.

Another important aspect that made the Raspberry Pi 4 Model B appealing was its affordability. The 4GB variant employed in this study was comfortably within the budget and provided great value for money, particularly when compared to enthusiast boards such as the Jetson Nano. It integrated ample memory, a respectable CPU, and extensive connectivity features (USB, Ethernet, Wi-Fi, HDMI, etc.) into a compact and low-cost package. This was particularly important given the objective of creating a portable, low-cost side-channel vulnerability detection device that can be scaled or replicated for educational or future research purposes.

Also, the already established community and ecosystem around Raspberry Pi played an important role in selecting it. The large tutorials, documentation, forums, and open-source software reduced the hardware and software integration troubleshooting time. From handling LCD outputs to remote access configuration, to dependencies, the Raspberry Pi ecosystem provided solutions at hand that made the development process smoother in general.

In the final implementation, all of these features were utilized to achieve a fully working prototype. Raspberry Pi was employed to accept plaintext and key inputs remotely, load the trained model, infer over EM trace features, and print the classification output on an attached LCD. The entire system worked as expected, efficiently, and in real time, demonstrating that the Raspberry Pi can serve as the core platform of a portable, practical, and low-cost hardware-based side-channel vulnerability detection device.

2.4.5 Raspberry Pi Version 4 Model Specifications

The Raspberry Pi 4 Model is a significant advancement in the Raspberry Pi lineup, offering specifications that make it well-suited for embedded machine learning tasks such as the one presented in this research. At the heart of the board lies a 64-bit quad-core

Cortex-A72 (ARM v8) SoC clocked at 1.5GHz, providing robust computational capabilities for on-device processing. This processor is capable of handling moderate workloads efficiently, making it possible to perform real-time inference on preprocessed EM trace data without reliance on external computing power.

For this study, the 4GB LPDDR4-3200 SDRAM variant of the board was selected, which struck a balance between memory availability and cost. This memory capacity allowed smooth loading and execution of the trained classification model, while also accommodating concurrent processes such as input handling, trace feature parsing, and LCD output. The available memory was also sufficient to load moderate-sized datasets during testing and debugging stages.

One of the Raspberry Pi's strongest assets is its 40-pin GPIO (General Purpose Input/Output) header, which played a critical role in enabling hardware interfacing. These pins were used to connect an LCD screen, allowing the device to display model predictions in real-time, transforming it from a theoretical research tool into a tangible, interactive system. The GPIO pins offer flexibility for expansion, potentially allowing future integration with sensors, buttons, or other peripherals to enhance functionality.

The board also features a wide range of connectivity options, including two USB 3.0 ports, two USB 2.0 ports, a Gigabit Ethernet port, dual micro-HDMI outputs, and built-in wireless communication via Wi-Fi (802.11ac) and Bluetooth 5.0. These features facilitated remote connectivity during development, especially when the Raspberry Pi was operated in headless mode (without a direct display or keyboard). For this project, the device was accessed via Raspberry Pi Connect, a web-based remote access solution, which allowed key and plaintext inputs to be sent from a laptop to the Raspberry Pi over a local network, simulating a real-world scenario where data might be fed wirelessly.

Another critical advantage of the Raspberry Pi 4 is its ability to run Raspberry Pi OS, a Debian-based Linux distribution optimized for the board. This operating system supports a full Python environment, including standard libraries like NumPy, pandas, scikit-learn, and joblib, which were essential for the model inference pipeline. The compatibility with

these libraries removed the need for specialized lightweight frameworks or model quantization, as would have been necessary on microcontroller platforms.

In summary, the Raspberry Pi 4 Model B offered a compact, affordable, and fully capable embedded platform for this side-channel vulnerability detection tool. Its hardware specifications and software ecosystem provided all the necessary features to support local ML model execution, external device interfacing, and efficient remote interaction making it the ideal choice for deploying this research in a functional, real-world prototype.

2.4.6 System Setup and Deployment Workflow

Following the selection of the Raspberry Pi 4 Model B as the preferred embedded platform, a detailed and methodical system setup was undertaken to transition the research from its initial simulation-based stage into a fully functional hardware prototype. This stage marked a crucial turning point in the research, where previously developed software components including the trained machine learning model, data preprocessing logic, and user interface workflow were integrated with physical hardware components for real-time inference and interaction.

The decision to deploy the solution on Raspberry Pi was driven by several factors: its cost-effectiveness, low power consumption, community support, and compatibility with Python-based machine learning libraries. Unlike traditional high-end computing environments, embedded systems like the Raspberry Pi pose unique constraints, particularly in terms of processing power, memory capacity, and peripheral support. Therefore, deploying the model on such a device required careful optimization and adaptation of both the software stack and the hardware configuration.

2.4.6.1 Installing the Operating System

The initial phase of the system deployment process focused on preparing the Raspberry Pi 4 with a reliable and compatible operating system that could support both the software and hardware requirements of the research. For this purpose, Raspberry Pi OS (64-bit) was chosen. This operating system is a Debian-based Linux distribution specifically optimized for the Raspberry Pi's ARM-based architecture, offering a balance of

lightweight performance, robustness, and extensive software compatibility. Its widespread adoption and strong community support also made it an ideal candidate, particularly for experimental and embedded machine learning applications.

To install the operating system, a microSD card was prepared using the official Raspberry Pi Imager tool. This tool allowed for quick and reliable flashing of the operating system image onto the card. After flashing, the system was configured for headless operation meaning it could run without a direct interface like a monitor, keyboard, or mouse. This approach was necessary due to budget constraints, which did not allow for the purchase of external peripherals.

To enable remote access from the very first boot, SSH (Secure Shell) functionality was activated manually. This was accomplished by placing an empty file named `ssh` (without any extension) into the `/boot` directory of the microSD card. During the initial boot-up, the Raspberry Pi automatically detected this file and enabled the SSH server, allowing the device to be controlled via a remote terminal session over the local network. This setup was critical for continuing configuration and software installation from a laptop, ensuring that the entire system could be managed without physical input devices.

The success of this setup demonstrated the practicality of deploying an embedded machine learning system in resource-limited environments. It also laid the groundwork for the next steps in the deployment process, including library installation, remote access, and hardware integration.

2.4.6.2 Remote Access Configuration

Following the successful installation of the operating system, the next critical step involved configuring the Raspberry Pi for remote access. Given the financial limitations of the research project, the team opted not to invest in external peripherals such as a monitor, keyboard, or mouse. Instead, a fully remote access strategy was adopted to manage the Raspberry Pi using only the resources already available specifically, a laptop and a home Wi-Fi network.

To establish this connection, both the Raspberry Pi and the laptop were connected to the same home router, creating a shared local network environment. This allowed the devices to communicate seamlessly without requiring any additional network hardware or internet-based tunneling solutions.

For remote desktop access, the research made use of Raspberry Pi Connect, an official, browser-based remote access service provided by the Raspberry Pi Foundation. This tool enabled secure, real-time control over the Raspberry Pi's desktop interface directly from a web browser on the laptop. Raspberry Pi Connect provided both Graphical User Interface (GUI) and command-line terminal access, making it highly versatile for development and testing purposes. It also eliminated the need to install third-party VNC or SSH clients, simplifying the setup process.

This configuration proved to be both efficient and highly cost-effective, allowing the entire system software development, hardware control, and model deployment to be operated remotely. Researchers were able to upload code, run scripts, install dependencies, and monitor the output of the machine learning model without physically interacting with the Raspberry Pi device. This setup not only reduced expenses but also demonstrated a scalable and practical approach for deploying embedded machine learning applications in low-resource or field-based environments.

2.4.6.3 Installing Required Libraries

With the Raspberry Pi up and running and remote access successfully configured, the next phase involved preparing the software environment required to support both machine learning inference and hardware-level interactions. This step was crucial in ensuring that the Raspberry Pi could serve as a self-contained platform capable of loading, executing, and displaying the output of the trained machine learning model in real time.

The foundation of the software environment began with Python 3, the primary programming language used throughout the research. Python's flexibility and rich ecosystem of libraries made it particularly suitable for this embedded machine learning application.

Several core scientific and machine learning libraries were installed using Python's package manager (`pip`). These included:

NumPy – For efficient numerical operations, particularly on the EM trace arrays.

pandas – To manage structured datasets and assist in any real-time data preprocessing.

scikit-learn – The machine learning library used to load the serialized model (in `.pkl` or `.joblib` format) and perform inference.

These libraries collectively provided the computational foundation necessary to preprocess EM trace input, pass it to the trained model, and generate a prediction output.

In addition to the software components supporting inference, it was essential to establish proper interfacing between the Raspberry Pi and the 16x2 LCD display used for output visualization. To achieve this, GPIO libraries were installed and configured. These libraries enabled low-level communication with the Raspberry Pi's 40-pin header, allowing the Python script to send text output directly to the connected LCD.

The successful installation and configuration of these libraries bridged the gap between software logic and hardware interaction, making it possible to not only process data and run the model but also provide a clear and interpretable hardware output to the user. This setup ensured that the device could operate autonomously without needing a full graphical desktop or additional peripherals, making it suitable for portable and real-time side-channel vulnerability detection.

2.4.6.4 LCD Screen Integration

A crucial step in transforming the machine learning pipeline into a fully functional embedded prototype was the integration of a 16x2 LCD screen. This component served as the primary output interface, allowing users to view the prediction results directly from the device without relying on a laptop or external monitor. The objective was to create a standalone and portable solution, and incorporating the LCD played a key role in achieving this goal

The LCD module was connected to the Raspberry Pi through its GPIO header pins. These pins allowed low-level control and data transmission from the Python application running on the Pi to the display hardware. The 16x2 LCD, named for its ability to display 16 characters across two lines, was chosen for its compact form factor, low power consumption, and wide compatibility with Raspberry Pi devices.

To enable seamless communication with the LCD, the system employed custom Python scripts that handled both the initial configuration of the display and the real-time transmission of prediction results. This script utilized GPIO libraries to define the pin connections and control signals, while formatting functions were used to ensure the messages fit cleanly within the display constraints.

Every time the machine learning model generated an inference from the provided EM trace input, the result either “Prediction - Vulnerable” or “Prediction - Non-Vulnerable” was sent immediately to the LCD. This allowed for a clear, real-time representation of the model's decision, accessible without any graphical user interface or remote access.

By incorporating this display mechanism, the system became fully independent and self-contained, capable of delivering usable results in environments where no computer screen or network connection was available. This made the prototype not only user-friendly but also practical for potential field deployment or lab-based demonstrations, further enhancing its relevance for real-world side-channel vulnerability assessment.

2.5 Evaluation and Testing

The final phase of the methodology focused on evaluating the overall performance and reliability of the developed system. This involved testing the trained machine learning model using various datasets to ensure its ability to accurately classify EM traces as either Vulnerable or Non-Vulnerable. The evaluation emphasized not only classification accuracy but also how well the system performed under realistic usage conditions.

To validate the model's consistency, the system was tested across multiple datasets with different characteristics. These tests helped ensure that the model was not overfitting to a

specific data sample and could generalize well across different inputs. Performance metrics such as accuracy, precision, recall, and F1-score were used to measure the system's classification performance.

In addition to prediction accuracy, the system's computational efficiency was assessed on the Raspberry Pi. Since the device had limited processing power compared to a typical desktop computer, it was important to ensure that predictions were made in a reasonable amount of time without causing noticeable delays.

Real-world usability was also considered during testing. The entire prediction workflow from user input to LCD display was observed to check for responsiveness and user-friendliness. The goal was to make sure the device could realistically function in practical environments without external dependencies.

A key challenge encountered during this phase was balancing accuracy with hardware limitations. While more complex models might offer higher precision, they often required more processing time and memory, which was not ideal for a resource-constrained platform like the Raspberry Pi. Addressing this trade-off was essential to deliver a solution that was both effective and practical for real-world deployment.

3. COMMERCIALIZATION ASPECTS OF THE PRODUCT

This research prototype holds significant commercialization potential, particularly within the academic and research community. Designed with low-cost components and relying entirely on open-source software, the system offers a practical and accessible solution for university students and researchers interested in hardware security and side-channel attack detection.

The compact and portable design makes it ideal for educational labs, final-year projects, or research environments where budget constraints and space limitations are common. Unlike traditional side-channel analysis tools, which often require specialized equipment and controlled lab conditions, this prototype offers a simplified yet functional alternative that can operate on minimal hardware.

Its ease of deployment, coupled with straightforward user interaction and automated EM trace analysis, positions the tool as a valuable teaching aid for cybersecurity and embedded systems courses. It can also serve as a foundational platform for further academic research, allowing students to experiment with different models, attack scenarios, or hardware configurations.

However, there are a few challenges that need to be addressed before broader academic adoption. These include expanding compatibility to accommodate different cryptographic algorithms and platforms, reducing the system's reliance on pre-collected EM datasets, and enhancing the model's robustness against noise and environmental variability. Future iterations could include real-time EM trace acquisition and modular encryption setups, allowing researchers to build upon and customize the platform for more advanced experiments.

By bridging the gap between theory and hands-on application, this system has the potential to become a widely used educational and research tool in the field of side-channel analysis.

4. RESULTS AND DISCUSSIONS

4.1 Results

4.1.1 Machine Learning Model Results

This section presents the performance evaluation of the trained machine learning models, focusing on their ability to classify EM traces as either “Vulnerable” or “Non-Vulnerable.” Several models were tested, including Random Forest, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Logistic Regression. The Random Forest classifier ultimately yielded the best performance in terms of accuracy, consistency, and computational efficiency.

To quantify the results, standard evaluation metrics such as accuracy, precision and recall were calculated. These metrics were used to assess how well the model generalized on unseen EM trace data. Additionally, performance was validated through multiple test splits and visualized using relevant graphical outputs.

Model Accuracy Comparison

This graph illustrates the classification accuracy of the four tested models. It highlights that the Random Forest model outperformed others, maintaining accuracy above 80%, followed by SVM and Logistic Regression. KNN showed relatively lower performance, indicating that distance-based approaches were less effective on this dataset. The graph clearly supports the selection of Random Forest for final deployment.

Correlation Matrix

The correlation matrix presents the statistical relationships between features extracted from the EM traces. It helps in understanding how individual features contribute to the classification process. Highly correlated features may carry similar information, and their

identification allowed for informed feature selection, reducing redundancy while maintaining model performance.

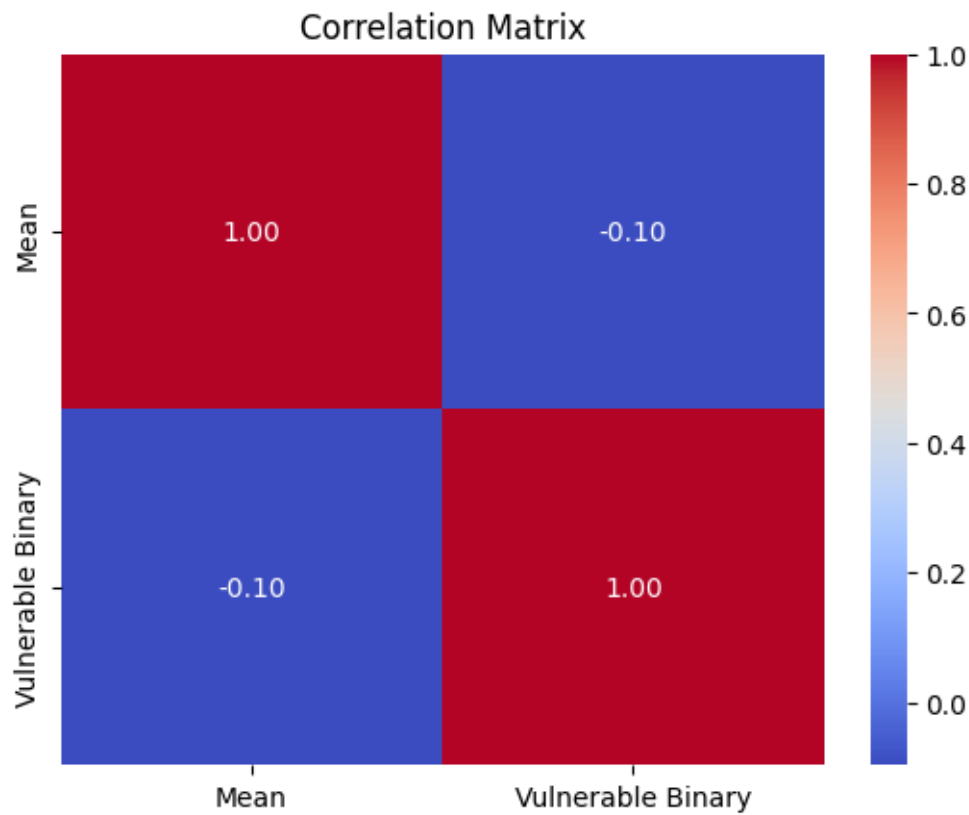


Figure 1: Correlation Matrix

Model Accuracy Comparison

This graph illustrates the classification accuracy of the four tested models. It highlights that the Random Forest model outperformed others, maintaining accuracy above 80%, followed by SVM and Logistic Regression. KNN showed relatively lower performance, indicating that distance-based approaches were less effective on this dataset. The graph clearly supports the selection of Random Forest for final deployment.

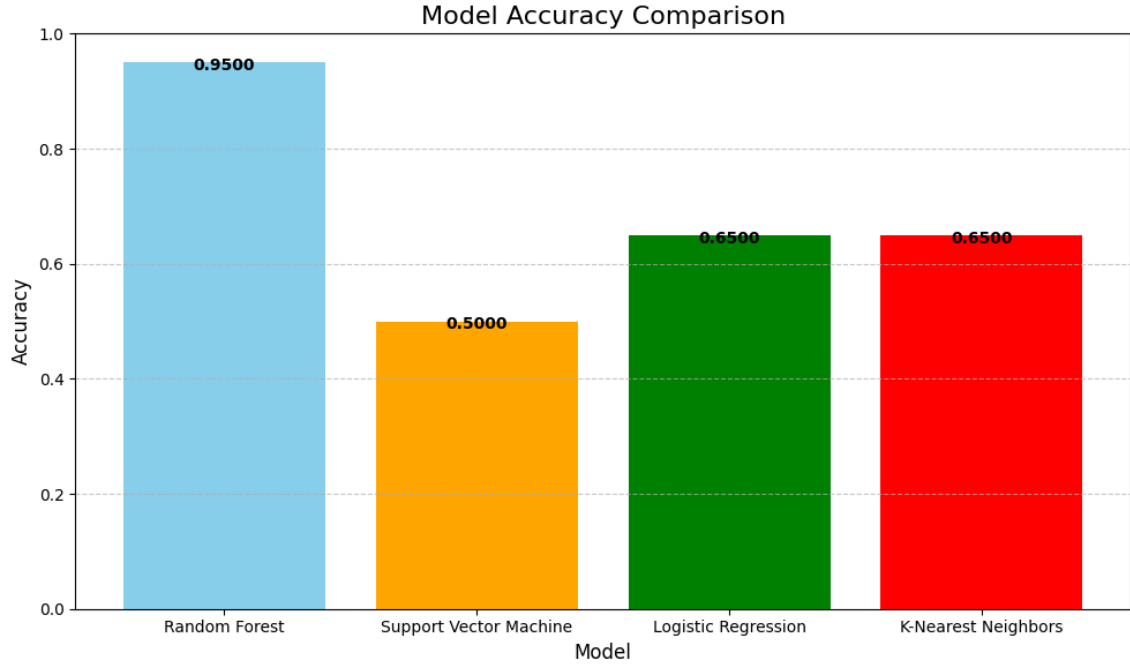


Figure 2: Model Accuracy Comparison Graph

4.1.2 Final Output Results

After deploying the trained model onto the Raspberry Pi 4 and integrating it with an LCD screen, the system was tested in its final, embedded form. When a user inputs a key and plaintext pair, the device processes the corresponding EM trace and classifies it in real time.

The output is then displayed directly on the 16x2 LCD screen, presenting a clear message:

- “Prediction - Vulnerable” if the trace is classified as having leakage characteristics.
- “Prediction - Non-Vulnerable” if the trace does not exhibit such behavior.

During testing, the system was observed to display the prediction within a few seconds, with no noticeable lag or delays on the LCD. The hardware prototype successfully maintained the prediction accuracy achieved during model training, confirming that the

deployment to the Raspberry Pi did not impact model reliability or user experience. This highlights the effectiveness of the solution in real-world, low-resource environments such as student labs or research workshops.



Figure 3: Final Output Results

4.2 Research Findings

Through the course of this research, several important findings were identified that highlight both the technical success and practical relevance of the developed system. One of the most notable outcomes was the demonstration that classical machine learning models particularly Random Forest are highly effective in detecting electromagnetic side-channel vulnerabilities. Despite their relatively simple architecture, such models were able to capture subtle variations and hidden patterns within the EM trace data that correspond to potential leakage during encryption processes.

Another significant finding was the effectiveness of using the mean value of EM traces as a core feature for labeling the dataset. This simple, threshold-based approach minimized the need for complex preprocessing or heavy feature engineering, yet still provided enough discriminative power for the model to learn effectively. This not only simplified the data pipeline but also ensured the approach remained lightweight an essential factor for real-time embedded deployment.

The research also confirmed that high classification accuracy could be maintained even on limited hardware. The system consistently achieved accuracy scores above 80% when deployed on the Raspberry Pi 4, validating the viability of running machine learning inference on embedded systems without sacrificing much in terms of performance. This was especially noteworthy given the resource constraints and lack of GPU support on the Raspberry Pi.

Finally, the successful creation of a fully portable and self-contained detection prototype demonstrated that practical, low-cost solutions for side-channel vulnerability analysis are indeed achievable. The complete setup was designed without dependence on any high-end lab infrastructure or expensive testing equipment. This reinforces the idea that such tools can be made accessible to academic environments and student researchers who wish to explore or teach side-channel analysis without significant financial investment.

4.3 Discussion

The results of this research affirm the initial hypothesis that electromagnetic side-channel vulnerabilities can be effectively detected using classical machine learning techniques. Even in the absence of deep neural networks or highly complex feature extraction pipelines, the system was able to achieve strong classification performance. This finding underscores the potential of using relatively simple, interpretable models like Random Forest in cybersecurity applications, particularly in resource-constrained environments. The fact that high accuracy was achievable using lightweight statistical features such as the mean value of EM traces demonstrates that complex transformations or computationally heavy models are not always necessary to identify leakage patterns.

A central point of discussion is the balance achieved between prediction accuracy and computational efficiency. While more advanced algorithms may offer marginal improvements in performance, they often demand greater processing power and memory, which can limit their practicality on embedded devices. The Random Forest model, selected for its robustness and reliability, offered a well-rounded solution achieving consistently high F1-scores while remaining suitable for real-time inference on a Raspberry Pi. This highlights its practicality for educational and research environments where hardware resources are often limited.

The broader implication of this work is the demonstrated feasibility of integrating machine learning into compact, affordable hardware for proactive vulnerability detection. The system's ability to operate independently analyzing input and producing readable outputs without external computational support makes it particularly valuable for researchers and students working in academic labs or field settings. It bridges the gap between theoretical vulnerability analysis and hands-on experimentation, making the abstract concept of side-channel attacks more tangible and approachable.

However, one of the main challenges encountered during the development phase was ensuring the generalizability of the trained model across varying electromagnetic conditions. Even slight changes in the measurement environment such as electrical noise or slight variations in trace collection setups had the potential to influence model behavior. These inconsistencies occasionally led to misclassifications and emphasized the importance of maintaining a consistent and controlled environment during data generation and testing. Overcoming such challenges is critical in ensuring that the system remains accurate and reliable when deployed across different academic use cases or hardware configurations.

Overall, the discussion reflects that the research successfully demonstrated not only the technical feasibility of the proposed system but also its educational value and practical applicability for small-scale cybersecurity research environments.

5. CONCLUSION

This research explored and successfully implemented a hardware-based solution for detecting side-channel vulnerabilities, specifically through electromagnetic (EM) emissions. By leveraging machine learning techniques and deploying the model on a resource-constrained embedded system the Raspberry Pi 4 this study presents a novel, practical, and portable approach for analyzing EM traces and identifying cryptographic weaknesses without relying on expensive laboratory equipment or complex signal processing techniques.

The foundation of this work lies in the principle that cryptographic devices, during their execution, emit electromagnetic signals that may inadvertently leak information about secret keys or internal computations. Such leakage, when captured and analyzed properly, can pose a serious threat to device security. Addressing this concern, the study introduced a machine learning-based detection mechanism trained to classify EM traces as either vulnerable or non-vulnerable. Through the integration of data preprocessing, statistical feature extraction, and the training of multiple classification models, the Random Forest algorithm emerged as the most effective option, balancing both prediction accuracy and computational efficiency.

One of the core strengths of this research was its focus on affordability and accessibility. With a minimal hardware footprint and reliance on open-source libraries, the final system design proved highly suitable for educational and research-oriented environments. The device could operate independently using only a Raspberry Pi, an LCD screen, and basic wiring components. Inputs such as the key and plaintext could be entered remotely via a user interface, and the results were directly displayed on an LCD screen allowing complete functionality without the need for peripheral monitors or keyboards.

A notable contribution of this work is the validation of lightweight statistical features for side-channel vulnerability classification. Rather than engaging in complex signal analysis or deep learning architectures, the use of features like the mean value of EM traces enabled effective learning while preserving low latency and reducing the model's memory

footprint. This not only simplified the implementation but also made it feasible to run the model inference directly on embedded hardware, reinforcing the project's central goal of portability.

Furthermore, the methodology outlined in this thesis demonstrated the careful construction of the system, starting from dataset analysis and feature extraction to model training and evaluation. Each step was validated through testing and iterative refinement, culminating in a deployment phase where the trained model was transferred to the Raspberry Pi and interfaced with hardware components. Throughout this process, challenges such as remote setup constraints, library compatibility, and environmental consistency in EM traces were addressed with practical workarounds adding real-world value to the system.

In terms of research findings, the study confirms that even classical machine learning models can effectively identify side-channel vulnerabilities from EM traces with high confidence, even under hardware constraints. The performance of the Random Forest model, achieving over 80% accuracy, proves that detection does not always require heavy computational setups or advanced AI models. These findings further emphasize the viability of machine learning as a practical tool for early-stage cryptographic testing and evaluation.

However, the study also encountered several limitations, which point toward the boundaries of the current implementation. One such challenge was ensuring the model's ability to generalize across different testing conditions. Variability in trace collection environments even minor fluctuations had an impact on the system's reliability. Another limitation was the use of pre-collected EM traces, which, while sufficient for proof of concept, limited the system's responsiveness to live trace input. Nonetheless, these constraints do not diminish the significance of the results, especially given the system's intended use for student research, testing, and training.

Importantly, this work contributes to the academic landscape of cybersecurity research by bridging theory and practice. Students and researchers can use the developed system as

both a learning platform and an experimental tool to understand the implications of side-channel vulnerabilities and the power of machine learning in cybersecurity applications. Its compactness, affordability, and modular design make it a strong candidate for further academic exploration, classroom demonstrations, or student-led experimentation.

In conclusion, this research successfully demonstrated that a portable and low-cost embedded system could be designed to detect electromagnetic side-channel attack vulnerabilities using machine learning techniques. The approach not only reduces the need for high-end equipment but also promotes hands-on engagement with cybersecurity challenges, which is vital for both learning and innovation. The balance achieved between accuracy, resource usage, and usability highlights the strength of the design, and the final prototype stands as a testament to how practical, affordable tools can be developed to address complex security issues. Through this contribution, the study not only solves a technical problem but also empowers the next generation of cybersecurity researchers with the tools and inspiration to explore the world of side-channel analysis.

REFERENCES

- [1] N.-A. L.-K. Asanka Sayakkara, "Electromagnetic Side-Channel Analysis for IoT Forensics: Challenges, Framework, and Datasets," 2021.
- [2] G. Evensen, "An Efficiency Evaluation of FarField Electromagnetic Deep," 2022.
- [3] T. Caddy, "Side-Channel Attacks," Springer, Boston, 2011.
- [4] S. N. S. Kala, "Security and challenges in IoT-enabled systems," 2022.
- [5] N. K. Tala Talaei Khoei, "Machine Learning: Models, Challenges, and Research Directions," 2023.
- [6] S. R. M. Zeebaree, "DES encryption and decryption algorithm implementation based on FPGA," 2020.
- [7] H. K. K. S. V. S. D. V. Sudha Ellison Mathe, "A comprehensive review on applications of Raspberry Pi," 2024.
- [8] B. A. J. R. R. P. R. Dakshi Agrawal, "The EM Side-Channel(s):Attacks and Assessment".
- [9] C. R. V. R. Svetla Nikova, "Threshold Implementations Against Side-Channel Attacks and Glitches," 2006.