ST. PÖLTEN UNIVERSITY
OF APPLIED SCIENCES

Informatik
& Security

fh
st.pölten

Long-Term Assessment of SRAM Physical Unclonable Functions on the ESP32

Analysing RTC SLOW Memory for Building Secure IoT Authentication Systems

Diploma thesis

for attainment of the academic degree of

Diplom-Ingenieur/in

submitted by

Christian Lepuschitz

51826356

in the

University Course Information Security at St. Pölten University of Applied Sciences

Supervision

Advisor: Dipl.-Ing. Dr. Henri Ruotsalainen

Assistance: -

Declaration

Title: Long-Term Assessment of SRAM Physical Unclonable Functions on the ESP32

Type of thesis: Diploma thesis

Author: Christian Lepuschitz

Student number: 51826356

I hereby affirm that

- I have written this thesis independently, have not used any sources or aids other than those indicated, and have not made use of any unauthorized assistance.
- I have not previously submitted this thesis topic to an assessor for evaluation or in any form as an examination paper, either in Austria or abroad.
- this thesis corresponds with the thesis assessed by the assessor.

I hereby declare that

☒ I have used a Large Language Model (LLM) to proofread the thesis.

☒ I have used a Large Language Model (LLM) to generate portions of the content of the thesis. I affirm that I have cited each generated sentence/paragraph with the original source. The LLM used is indicated by a footnote at the appropriate place.

☐ no Large Language Model (LLM) has been used for this work.

Date *Signature*

Kurzfassung

Durch die zunehmende Vernetzung von Geräten im Internet of Things (IoT)-Bereich steigen die Herausforderungen an deren Sicherheit. Trotz einer Vielzahl von Authentifizierungsmethoden basieren die meisten davon auf der Speicherung von Zugangsinformationen in nicht-flüchtigem Speicher. Im Falle einer physischen Kompromittierung von IoT-Geräten können solche Zugangsinformationen oftmals leicht gestohlen werden.

Static Random-Access Memory Physical Unclonable Functions (SRAM PUFs) bieten einen innovativen Ansatz zur Erzeugung von kryptografischen Schlüsseln. Sie basieren auf unregelmäßigen, submikroskopischen Herstellungsvariationen in SRAM-Speichern, die jedem produzierten Chip eine Art digitalen Fingerabdruck verleihen. Dieser Fingerabdruck ist für jeden Chip einzigartig, nicht klonbar und bietet daher ideale Voraussetzungen zur Authentifizierung von Geräten. Durch die Intra-Chip-Variabilität verändert sich dieser jedoch geringfügig zwischen jeder Verwendung, lässt sich durch externe Faktoren beeinflussen und weist zudem zeitliche Instabilität auf. Die Herausforderung besteht darin, trotz dieser Eigenschaften zuverlässige und sichere Authentifizierungssysteme zu entwickeln, ohne Zugangsinformationen am Gerät zu speichern.

In der vorliegenden Masterarbeit wird ein SRAM PUF basiertes Authentifizierungssystem für IoT-Geräte vorgestellt, welches speziell für den Einsatz am Mikrocontroller ESP32 getestet und entworfen wurde. Das entwickelte System unterscheidet sich von anderen in der Literatur vorgeschlagenen Authentifizierungssystemen primär in zwei Eigenschaften: Es nutzt ein Bit-Selection-Verfahren anstelle von Fehlerkorrekturverfahren und es setzt auf Application Programming Interface (API)-Keys anstelle von kryptografische Verfahren. Aufgrund des API-Key basierten Systems als zusätzlichen Faktor zu einer Mutual TLS (mTLS) basierten Authentifizierung, bietet dieses einen zusätzlichen Sicherheitsfaktor und lässt sich nahtlos in bestehende Authentifizierungssysteme einbinden.

Im Rahmen der Vorbereitungen wurden umfangreiche Langzeitmessungen durchgeführt, die Einflüsse verschiedener Umgebungstemperaturen und Spannungsversorgungen auf den SRAM evaluiert und die Unterschiede der uninitialisierten SRAM-Werte zwischen Microcontrollern derselben Serie sowie zwischen

verschiedenen Serien analysiert. Eine detaillierte Literaturrecherche bietet zudem einen umfassenden Überblick über das Internet of Things (IoT), PUF-Technologien mit speziellem Fokus auf SRAM PUFs sowie über Fehlerkorrekturverfahren.

Die Ergebnisse dieser Masterarbeit bieten eine solide Grundlage für zukünftige Forschungen und Entwicklungen im Bereich der SRAM PUF basierten Sicherheitsmechanismen.

Abstract

With the rise of Internet of Things (IoT) devices, security challenges have increased. While various authentication methods exist, most of them rely on storing access information in non-volatile memory. However, if an IoT device is physically compromised, these credentials may be easily stolen.

Static Random-Access Memory Physical Unclonable Functions (SRAM PUFs) provide an innovative approach to generating cryptographic keys. They rely on the irregular sub-microscopic variations in SRAM to produce a unique digital fingerprint for each chip, making it impossible to copy and ideal for device authentication. However, due to intra-chip-variability, the fingerprint changes slightly with each use and can also be affected by external factors and temporal instability. Overcoming these challenges entails creating a secure and reliable authentication system without storing any access information on the device.

This master's thesis introduces an SRAM PUF-based authentication system for IoT, tested and designed specifically for the ESP32 microcontroller. Differing from other authentication systems proposed in the literature, this system uses a bit selection method instead of error correction methods and relies on Application Programming Interface (API) keys rather than cryptographic methods. The use of the API key-based system as an additional factor to Mutual TLS (mTLS) authentication provides an additional security factor and can be seamlessly integrated into existing authentication systems.

As part of the preparations, extensive long-term measurements were carried out. The influences of different ambient temperatures and power supplies on the SRAM were evaluated, and the differences in the uninitialised SRAM values between microcontrollers of the same series and between different series were analysed. A detailed literature review also provides a comprehensive overview of the IoT and Physical Unclonable Function (PUF) technologies, specifically focusing on SRAM PUFs and error correction methods.

The results of this master thesis provide a solid foundation for future research and development in SRAM PUF-based security mechanisms.

Contents

1. Introduction

Static Random-Access Memory Physical Unclonable Functions (SRAM PUFs) offer a promising approach to secure authentication of Internet of Things (IoT) devices. They are based on the uniqueness of SRAM memories, which results from sub-microscopic manufacturing variations. As soon as an SRAM memory receives power, each cell in the SRAM assumes a state of either 0 or 1. Theoretically, the probability for both cases is 50%, but due to minute production differences at the atomic level, certain cells tend to be 0, and other cells tend to be 1. Other cells are volatile and change their value whenever switched on. If a large part or the entire SRAM memory is now read out in an uninitialised state, this results in a pattern of 0's and 1's, which is very similar for every SRAM memory with every measurement but completely different to the patterns of other SRAM memories. These properties cannot be technically copied, which is why they are considered a Physical Unclonable Function (PUF). Due to their properties, SRAM PUFs offer good prerequisites for creating a hardware-based digital fingerprint and, thus, uniquely identifying and authenticating devices.

SRAM PUFs are a popular research topic and are used in various industries such as aerospace, defence, medical equipment, and IoT [1]. Research in this field mainly concentrates on analysing dedicated SRAM memories, developing methods for costly Field Programmable Gate Arrays (FPGAs), and designing cryptographic authentication methods that necessitate special server software. In recent years, there has also been increasing research into the feasibility of implementation for standard microcontrollers in IoT devices. Nevertheless, there still needs to be more easy-to-implement solutions for the masses. This master's thesis aims to close this gap and develop an easy-to-implement procedure that can be seamlessly integrated into existing authentication systems without server-side adaptations.

1.1. Contribution

As part of this work, extensive measurements were carried out on several ESP32 microcontrollers, confirming them as good candidates for an SRAM PUF-based authentication method. The effects of silicone ageing on several microcontrollers were analysed as part of automated long-term measurements. Subsequent manual tests analysed the influence of external factors, such as the ambient temperature and a variable power supply, on the stability of the PUFs. Furthermore, the distribution of 0's and 1's in each tested ESP32's memory was evaluated, and the differences between all microcontrollers used were evaluated. The results provide an in-depth insight into the characteristics of the SRAM used and can be used by future research as a basis for SRAM PUF procedures on the ESP32.
In addition, a simple authentication system was proposed that can derive a large number of different API keys using SRAM PUF. This allows IoT devices to be authenticated to server systems without storing access information locally. Suppose attackers manage to clone the firmware and install it on identical microcontrollers. In that case, they cannot authenticate themselves, as the SRAM memories differ from the original system at an atomic level.

1.2. Thesis Outline

This thesis is divided into several chapters, which can be roughly subdivided into a theoretical and a practical part. Chapters 1 to 3 cover the theoretical part, while chapters 4 to 7 focus on the practical part.

1. **Introduction:** provides a general overview of the research subject and introduces the topic in more detail
2. **Prerequisites:** This chapter deals with the basics necessary to understand the present work. Specifically, it is dedicated to the Internet of Things (IoT), Physical Unclonable Functions (PUFs), Error Correcting Codes (ECCs) Static Random-Access Memory (SRAM).
3. **Related Work:** This chapter provides an overview of research in this area and presents some selected papers in more detail. The papers mentioned serve as a basis for the selected test procedures and the design of the proposed authentication system.
4. **Methodology:** This chapter explains the structure of this master thesis in detail. It explains how the selection of the microcontrollers took place, how the measurements and subsequent analyses were carried out, and based on which decisions the proposed authentication system is now available in its current form.

5. **Setup:** This chapter describes the setup of the development environment, firmware writing for microcontrollers, and data storage.
6. **Measuring and Analysing the SRAM:** This chapter is dedicated to the structure of the measurements and the analysis of the SRAM. It describes how the microcontroller was programmed, how the SRAM was read out, how the data was stored during the long-term measurements and how the collected data was finally analysed.
7. **Implementing an Authentication System:** This chapter covers the technical implementation of the authentication system. It explains how the bits are selected, how a key is derived from them, what a simple HTTP authentication server might look like and how the devices are authenticated via an encrypted connection.
8. **Conclusion:** At the end of the paper, the results of the analyses are briefly summarised, and the authentication system is explained again.
9. **Appendix Visualisations:** This part contains a part of the measurements carried out and the source code written to evaluate the measurements and the authentication system.

1.3. Limitations

The authentication system developed was tested for functionality but not analysed in terms of security. For this reason, it should only be used for testing purposes or for further research. In Section 8.1, some potential research topics are suggested that could enhance the security of this system.

2. Prerequisites

This chapter provides a comprehensive overview of the following topics: Internet of Things (IoT) [Section 2.1], Physical Unclonable Functions (PUFs) [Section 2.2], Error Correcting Codes (ECCs) [Section 2.3], and Static Random-Access Memory (SRAM) [Section 2.4].

2.1. Internet of Things (IoT)

The Internet of Things (IoT) refers to a network of physical objectsdevices, vehicles, appliances, buildings, and other itemsembedded with sensors, software, and other technologies to connect and exchange data with other devices and systems over the internet [2]. These devices, often called "smart" devices, can collect, send, and act on data they acquire from their environments [3]. IoT applications range from consumer products like smart home devices (thermostats, lighting systems, security cameras) and wearable technology to industrial and infrastructure applications such as smart cities, smart grids, and advanced manufacturing. [1]
In IoT, several options exist to transmit data, including Wi-Fi, Bluetooth, satellite and Long Range Wide Area Network (LoRaWAN) [4] [5]. With the use of internet connectivity and wireless transmission technologies, various sources of information, such as sensors, smartphones, and cars, are becoming increasingly interconnected. The number of internet-enabled devices is - seemingly exponentially - increasing [6].

2.1.1. Use Cases

Intelligent networking and data analysis are enabling IoT to revolutionize the way we work in a wide variety of areas, making work more efficient and autonomous.

In **agriculture**, IoT helps develop the supply and growth of crops by obtaining information about the environment through various sensors. In the **food** industry, air quality and oxygen sensors ensure the safety

[1] Generated by GPT-4

of workers. In the meat processing industry, IoT monitors ozone levels to ensure the safety and quality of products.

In **healthcare**, IoT systems monitor the health status of patients and generate alerts if necessary. Nursing staff can thus use networked systems to remotely monitor patients with a wide range of disorders such as diabetes, dementia, Alzheimer's and others.

In urban areas, IoT systems are used to **optimize traffic flow**. For example, city buses are monitored using GPS and time information to provide a city-wide overview of current traffic conditions and forecasts of arrival times, transit times and congestion routes on digital maps.

Industry, transport and business are not the only areas in which IoT systems are used; they are also used in a wide range of **smart home** devices. Through a combination of sensors and intelligent systems, everyday objects are connected in a network to communicate and solve problems without human intervention. [6]

2.1.2. Communication Protocols

Depending on their type and use, IoT systems communicate with various protocols. The choice of the network protocol used depends on various factors, such as the location where the IoT system is operated, the amount of data to be transmitted, power consumption (especially in battery operation), the size of the network and the available frequency bands (which may differ from country to country). Equally important is the distance over which the data is transmitted. Network protocols such as Sigfox or Cellular, used in smart grid systems or for controlling street lighting, are particularly suitable for long-distance data transmission (up to 50 km). Data rates for protocols designed for long distances, such as Sigfox, are generally rather low (100-600bps), and the power consumption is considerably higher [7].

Network protocols with higher transmission rates but shorter distances include Bluetooth, ZigBee and IPv6 over Low power Wireless Personal Area Network (6LoWPAN). These three, especially the latter, are characterized by very low power consumption and thus enable battery operating times of 1-2 years [7].
Although Near Field Communication (NFC) and Radio Frequency Identification (RFID) are used in IoT systems, they only work over very short distances and can only transmit a small amount of data. RFID tags can be used either actively or passively (with or without an embedded battery). However, the data stored

on a RFID tag is static and cannot be used for transmitting changing data, such as measurements. Although NFC works similarly to RFID, it is designed for even shorter distances and uses a two-way communication system [7].

Which network protocol is most suitable for an IoT system, consequently, depends on the following factors

- Available frequency band
- Energy efficiency
- Required data rate
- Required range
- Security (encryption)
- Scalability

Just like network protocols, application protocols have their advantages and disadvantages. In the following, however, only Internet Protocol (IP)-based protocols are described. The most commonly used IoT protocols include Constrained Application Protocol (CoAP), Extensible Messaging and Presence Protocol (XMPP), representational state transfer (REST)ful Hypertext Transfer Protocol (HTTP), Message Queuing Telemetry Transport (MQTT), WebSocket, Advanced Message Queuing Protocol (AMQP) and Data Distribution Service (DDS). Contrary to the Transmission Control Protocol (TCP)-based protocols, CoAP and DDS are based on User Datagram Protocol (UDP), making them the most lightweight protocols [8].

[8] outlines the advantages and disadvantages of the protocols mentioned as follows:

CoAP:

- **Advantages**: It supports multicast, has low overhead, and minimizes the complexity of mapping with HTTP.
- **Disadvantages**: It does not enable communication layer security, and there are few existing libraries and solution support.

XMPP:

- **Advantages**: Flexibility of communication models, low latency, real-time, easy to understand, and extend; any XMPP server can be isolated.
- **Disadvantages**: Heavy data overhead and, therefore, not suitable for embedded IoT applications.

RESTful HTTP:

- **Advantages**: Easy application maintenance and no client state management on the server.
- **Disadvantages**: The client must store all data needed to perform a query locally, there is no error handling, and it is difficult to extend.

MQTT:

- **Advantages**: It is useful for remote site connections, is easy to implement, has a small code footprint, is lightweight, and uses an asymmetric client-server relationship.
- **Disadvantages**: Basic message queuing implementations, no error-handling, does not address connection security, and extensions are hard to add.

WebSocket:

- **Advantages**: Simplifies web communication and co-network compatibility and offers connection management.
- **Disadvantages**: It has specific hardware requirements and does not offer useful open-source implementations targeted at embedded systems.

AMQP:

- **Advantages**: It is an International Organization for Standardization (ISO) standard that provides complex message queuing implementations, high routing reliability, and security, is easily extensible, and has a symmetric client-server relationship.
- **Disadvantages**: It has a larger packet size than other protocols and does not support Last Value Queue (LVQ).

DDS:

- **Advantages**: It enables real-time monitoring of quality of service, features a decentralized architecture, and allows dynamic detection of senders and subscribers.

2.1.3. Authentication Methods in IoT

The authentication procedures for IoT systems differ from traditional authentication methods in that device authentication is used instead of user authentication. This means that the identity of a user is not checked interactively, but the device itself confirms its identity to other participants or servers. Although device authentication is also used in classic IT systems, there are some particular difficulties regarding IoT authentication. These include limited resources, connection security (often wireless) and scalability.

In the following, the authentication methods such as OTP, Certificate Based Authentication (CBA), Identity Management (IDM), Biometry and Physical Unclonable Function (PUF) are briefly explained based on the results of a literature search in [9]:

- **One Time Password (OTP)** based authentication methods [10] rely on pairing techniques facilitated by elliptic curves. The generation occurs in four phases: setup, extraction, generation and validation. In the first phase, a pair of prime numbers is generated, and in the second phase, IoT applications and devices register with the Private Key Generator (PKG) and receive private and public keys. In the third phase, the application requests data from the IoT device, and the PKG automatically generates the key of the requesting device. During validation in the fourth phase, the application and the IoT device exchange data via the OTP. The IoT device checks whether the OTP originates from the application and performs the requested task. This method for generating the OTP is based on the Lamport algorithm. [9]

- **Certificate Based Authentications (CBAs)** offer a secure approach to authenticating IoT devices. Certificates are issued to IoT devices by a central certification authority, as is common in a classic Public Key Infrastructure (PKI) infrastructure. Entities that issue certificates and entities that are responsible for authentication are strictly separated. The Datagram Transport Layer Security (DTLS) handshake protocol is used to authenticate IoT devices on the server, as proposed in [11]. However, the DTLS handshake protocol is only available without restrictions on IoT systems with sufficient Random-Access Memory (RAM) and Read-Only Memory (ROM). These resources are often limited on IoT devices where the DTLS protocol is, therefore, not entirely usable. [9]

- **Identity Management (IDM)** based procedures [12] provide authorization and authentication for IoT users. Using a key-based authentication method, it provides single sign-on to IoT devices. An

Identity Provider (idP) manages users' identity and authentication characteristics and thus provides the required credentials for systems. This system enables a non-interactive login and provides identity checks based on private keys, which have proven to be more secure than passwords. [9]

- **Biometric** authentication systems use a person's unique physical characteristics, such as their iris, fingerprint, heartbeat, or voice, to verify their identity. Although these systems are relatively new to the field of IoT, they have the potential to protect personal IoT devices from theft. This is achieved by allowing only the owner, who has confirmed their identity using biometric features, to access the device. Biometric authentication is particularly useful for devices like personal fitness trackers with limited input options. Moreover, it can be used for authentication in other areas, such as unlocking/locking cars, homes or bank accounts. [13]

An alternative approach to the classic approaches mentioned above is using Physical Unclonable Functions. This concept is explored in detail in a dedicated chapter of this thesis. In essence, the PUF approach generates a key from a device's unique physical characteristics, safeguarding against the risks of compromise through counterfeiting or cloning of IoT systems.

2.1.4. Security Problems

IoT devices face a variety of different security issues. This chapter provides a detailed overview of security issues in IoT systems and explains methods to address these security threats. This chapter is based on findings and recommendations from [14], which provides a comprehensive analysis of the security landscape in the IoT sector.

- **Deficient physical security:** IoT devices are primarily operated in unmonitored environments, making them susceptible to unauthorized access and control by attackers with minimal effort. This vulnerability allows for inflicting physical damage, exposing implemented cryptographic systems, or replicating the device's firmware.

- **Insufficient energy harvesting:** By default, IoT systems can not renew their energy. Attackers can carry out targeted attacks on IoT systems to consume the stored energy and render the systems unusable for regular operation.

- **Inadequate authentication:** Due to the limited resources of IoT systems, implementing secure authentication mechanisms poses a significant challenge. Attackers can attack weak authentication systems, thus jeopardising data integrity. The authentication keys used are at permanent risk of being lost, destroyed or damaged. If the keys are not securely stored or transmitted, authentication systems may not be sufficient to ensure data integrity.

- **Improper encryption:** Due to resource limitations in IoT systems, encryption mechanisms are not always implemented with sufficient robustness. It is often possible for attackers to bypass the implemented encryption mechanisms and gain access to sensitive information.

- **Unnecessary open ports:** Many IoT devices have open ports that are not intended to be accessible from the internet. These include, for example, services used to manage or maintain these devices. These services are often not adequately protected, making them vulnerable to attacks.

- **Insufficient access control:** Numerous manufacturers of IoT systems have not implemented strong password policies and deliver devices with default credentials. Users often do not change these default credentials, providing a target for various attacks. In addition, users with elevated rights often use the default credentials.

- **Improper patch management capabilities:** Regular security updates and patches should be provided to ensure the secure operation of IoT systems. In many cases, manufacturers do not regularly provide security updates, or there are no automatic update mechanisms to keep the devices up to date. Even if automatic update mechanisms are in place, integrity checks are often not in place, making IoT devices vulnerable to fake, malicious updates.

- **Weak programming practices:** Many products in the IoT sector do not adhere to common programming and security practices, which can lead to vulnerabilities. For example, numerous products are delivered with known vulnerabilities, such as backdoors and root users as the main access point. Furthermore, Secure Socket Layer (SSL) is not always utilised, meaning that data is transmitted in plain text and can be read by external parties. Attackers can exploit these vulnerabilities to obtain and modify data and gain unauthorised access to systems, among other malicious activities.

- **Insufficient audit mechanisms:** Logging and auditing procedures are usually not used in the IoT environment for various reasons. As a result, detection and response in the event of an attack are difficult or even impossible.

2.1.5. ESP32 Microcontroller

The ESP32 from Espressif Systems [15] is a versatile and powerful microcontroller used in various applications like smart homes, wearable electronics and industrial automation. It offers a multitude of features, including Wi-Fi and Bluetooth connectivity, a dual-core processor, a wide range of General Purpose Input/Output (GPIO) pins, and support for various communication modes, making it ideal for IoT projects. It can withstand operating temperatures ranging from -40°C to +125°C degrees (depending on the exact model) and has a very low power consumption (10µA in deep sleep mode), perfect for use in remote locations without a continuous power supply.

Besides numerous standard features, the ESP32 also has dedicated security features. These include [15]

- Secure boot
- Flash encryption
- 1024-bit One Time Password (OTP), up to 768-bit for customers
- Cryptographic hardware acceleration:
  - Advanced Encryption Standard (AES)
  - Hash (Secure Hash Algorithm 2 (SHA-2))
  - Rivest-Shamir-Adleman (RSA)
  - Elliptic Curve Cryptography (ECC)
  - Random Number Generator (RNG)

Due to the wide range of possible applications, the economically attractive price, the widespread use in a wide range of products and the presence of safety functions, this microcontroller was chosen for the subject of this master's thesis.

2.2. Physical Unclonable Functions

Today's Information Technology (IT) systems use digital keys to protect confidential information and identities and guarantee authenticity. These digital keys form the basis for the cryptographic protocols used today. While the cryptographic protocols are public and can be viewed by anyone, the digital key (private key) must be kept secret. In traditional IT systems, the digital keys are usually stored in Non Volatile Memories (NVM) (such as those used in Trusted Platform Modules (TPMs)). The secure storage of cryptographic keys over a prolonged period of time poses a significant difficulty, especially if NVMs do not have a dedicated protection mechanism. The keys could, therefore, be read and the security of a system could be jeopardized.

In addition to the storage of cryptographic key material in NVMs, there are other options, such as the use of Hardware Security Modules (HSMs) or storage in the cloud [16]. An alternative approach to storage is offered by so-called Physical Unclonable Functions (PUFs). PUFs use inherent, random variations in the hardware resulting from minimal production deviations to generate secret key material [17]. These minimal variations vary from device to device but are always unique, similar to fingerprints. Although it is possible to measure the random variations, creating an exact physical copy remains impractical [17].

The use of PUFs is usually very cost-effective and does not affect the function of Integrated Circuits (ICs).

PUFs must meet the following requirements [18]:

- robust (they should not change over time)
- unique (they must be different from device to device)
- easy to test
- difficult or impossible to replicate
- difficult or impossible to predict.

2.2.1. Strong and Weak PUFs

The strength of PUFs is usually divided into the categories **strong** and **weak**. Weak and strong PUFs differ in the number of Challenge Response Pairs (CRPs) that can be generated by a single device and the ratio in which they increase with increasing device size. Weak PUFs scale worse (linear or polynomial) than strong PUFs (exponential).

If an attacker gains physical access to a PUF, they can read all possible CRPs, while it is not possible to copy the physical PUF itself. With a strong PUF, on the other hand, it is not possible for an attacker to read all CRPs. Even if an attacker has access to the PUF and reads out many CRPs, the probability that he will later be able to solve a challenge with it is negligible. [18]

2.2.2. Classification of PUFs

There are many different approaches to implementing PUFs; [18] divides them into different levels using an organic system, including application, randomness, source, family and concept. Applications are divided into "All Electronic", where the PUF is directly integrated into the electronics, and "Hybrid", where external sources such as external light sources may be used.

Implicit (Intrinsic) and Explicit (Extrinsic) PUFs

The source of randomness (randomness source) can be categorized into two types: implicit (intrinsic) and explicit (extrinsic). PUFs with implicit evaluation possess inherent randomness, which is evaluated internally. The mechanisms for validating the PUF are, therefore, embedded directly into the device. Further processing, such as hashing, can occur without the PUF response being externally accessible. Implicit assessments are, therefore, more resistant to MITM and side-channel attacks and tend to be more accurate, easier to use and less susceptible to interference from attackers.
In contrast, explicit PUFs do not have an integrated source of randomness integrated into the device but are located externally. The validation of the PUF (after the measurement) is therefore conducted externally to the device. [18]

2.2.3. Types and Implementations

PUFs are divided into different families, categorized based on their functionality [18]. The categories are:

- **Racetrack** *(analyses system latency by measuring signal completion time through wiring or components. They operate in the time domain and are similar to transient/glitch PUFs, but they differ in signal propagation examination)* [19] [20] [21]
- **Transient/Glitch** *(evaluate temporary signals emitted by circuits. Combining several ring oscillators and XORing them generates distinctive patterns, which are used to identify transient errors in signal inputs or correct any temporary glitches in intricate circuit functions)* [22] [23]

- **Direct Characterization** *(involves direct characterisation of electronic components without additional fabrication steps. Properties such as current response to voltage, capacitance, or circuit interconnect are examined)* [24] [25] [26] [27]
- **Volatile Memory** *(utilise unit cell properties to derive responses, including examining SRAM and DRAM reactions)* [28] [29] [30] [31]
- **Non Volatile Memory** *(store data without constant power using resettable cell property. They create random patterns with modified writing signals, acting as a random number generator and storage for a random key)* [32] [33] [34]
- **Radio Frequency** *(assess radio-frequency radiation interaction with objects for randomness. Some RFID systems use all-electronic PUFs without radiofrequency evaluation)* [35] [36]
- **Optical** *(uses visible light interacting with a random microstructure to create unpredictable patterns. The original form involves shining a laser through a plate with refractive particles for pattern analysis)* [37] [38] [39] [40] [41]
- **Magnetic** *(explores the magnetic field surrounding an object to investigate the inherent randomness during manufacturing)*[42]

Some of these categories are further subdivided into Time Domain PUFs (including Racetrack and Transient/Glitch), Memory Cell PUFs (including Volatile Memory and Non Volatile Memory) and Optical PUFs (Implicit and Explicit). [18]

2.2.4. PUF Implementation: A Categorized Overview

An overview of various PUF implementations, including assignment to applications, randomness source and family, was taken from [18] and split into two different tables: **All Electronic** (Table 2.1) and **Hybrid** (Table 2.2).

Application	Randomness Source	Family	Concept
All Electronic	Implicit	Racetrack	Ring Oscillator PUF
All Electronic	Implicit	Racetrack	Arbiter PUF
All Electronic	Implicit	Racetrack	Clock PUF
All Electronic	Implicit	Transient/Glitch	Glitch PUF
All Electronic	Implicit	Transient/Glitch	Transient Effect Ring Oscillator (TERO) PUF
All Electronic	Implicit	Direct Characterisation	Threshold Voltage (TV)-PUF
All Electronic	Implicit	Direct Characterisation	Power Distro. PUF
All Electronic	Implicit	Direct Characterisation	Cellular Neural Network (CNN) PUF
All Electronic	Implicit	Direct Characterisation	Vertical Interconnect Access (VIA) PUF
All Electronic	Implicit	Direct Characterisation	QUALPUF
All Electronic	Implicit	Volatile Memory	SRAM PUF
All Electronic	Implicit	Volatile Memory	Bistable Ring PUF
All Electronic	Implicit	Volatile Memory	MEmory Cell-based Chip Authentication (MECCA) PUF
All Electronic	Implicit	Volatile Memory	SRAM Failure PUF
All Electronic	Implicit	Volatile Memory	Dynamic Random Access Memory (DRAM) PUF
All Electronic	Implicit	Volatile Memory	Rowhammer PUF
All Electronic	Explicit	Non Volatile Memory	Memristor PUF
All Electronic	Explicit	Non Volatile Memory	Phase change key generator (PCKGen)
All Electronic	Explicit	Non Volatile Memory	Spin-Transfer-Torque Magnetic RAM (STT-MRAM) PUF
All Electronic	Explicit	Direct Characterisation	Accoustical PUF
All Electronic	Explicit	Direct Characterisation	Coating PUF
All Electronic	Explicit	Direct Characterisation	Super High Information Content (SHIC) PUF
All Electronic	Explicit	Direct Characterisation	Micro-Electrico-Mechanical System (MEMS) PUF
All Electronic	Explicit	Direct Characterisation	Carbon Nanotube (CN) PUF
All Electronic	Explicit	Direct Characterisation	Board PUF
All Electronic	Explicit	Direct Characterisation	Quantum Electronic PUF (Q-EPUF)
All Electronic	Explicit	Direct Characterisation	Self-Assembly PUF
All Electronic	Explicit	Direct Characterisation	Nano-Electrico-Mechanical System (NEMS) PUF

Table 2.1.: All Electronic PUF implementations

Application	Randomness Source	Family	Concept
Hybrid	Explicit	Radio Frequency	Radio-Frequency DNA (RF-DNA) PUF
Hybrid	Explicit	Radio Frequency	LC (Inductor [symbolized by L] - Capacitor) PUF
Hybrid	Explicit	Optical	Optical PUF
Hybrid	Explicit	Optical	Phosphor PUF
Hybrid	Explicit	Optical	Nanowire Distro. PUF
Hybrid	Explicit	Optical	Optical Fibre PUF
Hybrid	Explicit	Optical	Nanoparticle Distro. PUF
Hybrid	Explicit	Optical	Liquid Crystal PUF
Hybrid	Explicit	Optical	Quantum Optical PUF (Q-OPUF)
Hybrid	Explicit	Optical	Monolayer Depo. PUF
Hybrid	Implicit	Optical	Paper PUF
Hybrid	Implicit	Optical	Compact Disk (CD) PUF
Hybrid	Implicit	Magnetic	Magnetic PUF

Table 2.2.: Hybrid PUF implementations

According to [18], the most important representatives of the PUFs mentioned above include

- **Ring Oscillator PUFs** (Intrinsic / Racetrack)
- **Arbiter PUF** (Intrinsic / Racetrack)
- **Threshold Voltage (TV)-PUF** (Intrinsic / Direct Characterization)
- **Power Distro. PUF** (Intrinsic / Direct Characterization)
- **Vertical Interconnect Access (VIA) PUF** (Intrinsic / Direct Characterization)
- **Static Random-Access Memory (SRAM) PUF** (Intrinsic / Volatile Memory)
- **Quantum Optical PUF (Q-OPUF)** (Explicit / Optical)

2.3. Error Correcting Codes

Wherever digital information is processed, stored or transmitted, errors can occur. Individual bits can change their value from 0 to 1 or vice versa, rendering data unusable or compromising data integrity. Error correction codes were developed to correct these errors and improve the reliability of data transmission and storage [43].

As early as the 1940s, Claude Shannon [44] showed in a theorem that nearly error-free communication is possible via a noisy channel. Redundancy is added to the transmitted data to ensure errors can be detected and corrected at the receiving end. This means the original message can be read even though it was corrupted during transmission.

Errors can occur not only during the transmission of information via noisy channels but also during the storage of information on a storage medium. In this case, errors are caused by hardware faults, electrical interference, magnetic interference or environmental conditions. As with data transmission, errors are recognised and corrected here by adding redundancy information with the help of error correction codes. [45]

2.3.1. Applications of Error Correction

Among the first users of error correction codes were the then-emerging computer industry and the phone industry. Later, this technology also found increasing use with the advent of space exploration. The payload of early rockets was sometimes limited due to the restricted lifting power. The radius and microwave transmitters, therefore, had comparatively poor performance. Error correction codes were used to solve the problem of the poor quality of the signals received. The National Aeronautics and Space Administration (NASA) used a wide variety of error correction codes for all space missions, including the Reed-Muller code for the Mariner Spacecraft missions between 1969 and 1977, the Golay code for the Voyager 2 flight to Saturn and Jupiter and the Reed-Solomon code for a Voyager mission. Despite efforts to find a universally applicable error correction code, it has been shown that certain correction codes are more suitable for certain applications than others. [46]

Other examples of the use of error correction methods are

- **Compact Disks (CDs):** Compact discs are used to store data and mainly use the Reed-Solomon code for error correction, although some proprietary methods are also used. [46]
- **Memory modules:** Bits are defined in memory modules by electrical charge stored in "wells". The stored bit becomes faulty if a well unintentionally loses or gains its charge. Such errors occur in memory modules and must be corrected using error correction codes, in this case, often Hamming codes. [45]
- **WiFi and Long-Term Evolution (LTE):** With both standards, data is transmitted at a high data rate via disturbed channels. Low-Density Parity-Check (LDPC) codes are primarily used for error

correction. [47]

- **Hard Drive Disks (HDDs) and Solid State Drives (SSDs):** SSD controllers have error detection and correction functions to ensure reliability during data storage. The methods used for NAND flash memory are Hamming Code, Reed-Solomon (RS) Code, Bose-Chaudhuri-Hocquenghem Code (BCH) and Low-Density Parity-Check (LDPC). [48]

2.3.2. Error Correction in Physical Unclonable Functions

With physical unclonable functions, codes are generated from random process variations, which can be used for subsequent authentication processes. However, the values extracted from the PUF are unstable and can vary each time they are used. Due to this variance, the PUF responses cannot be used directly but must first be converted into a usable form. Efficient error correction and key derivation are, therefore, prerequisites for generating secure cryptographic keys using PUFs. Most methods can be divided into linear schemes and pointer-based schemes.
In order to be able to generate reliable keys from the always different PUF responses, helper data must be generated, with the help of which errors can later be detected and corrected using error correction codes. The helper data must be stored permanently on an on-chip NVM or a remote server. In both cases, the helper data's integrity must be ensured to be secure against data manipulation or side-channel attacks. [49]

In order to increase the quality of PUFs and simplify error correction, the PUF response actually used can be preselected. When reading out a PUF response, certain sequences may be more stable than others. If the size of the available stable sequences is sufficient, the effectively used PUF value can be reduced to the stable portion, meaning fewer errors must be corrected later, increasing the probability of more reliable data. [49]

2.4. Static Random Access Memory (SRAM)

Static Random-Access Memory (SRAM) is a form of computer memory used in microprocessors, general computing applications and electronics. Data is stored statically in SRAM memories and, therefore, does not need to be refreshed, as with Dynamic Random Access Memory (DRAM). Despite this, the data stored in it is volatile, i.e. if the power supply is interrupted, the stored data is lost. Compared to DRAM, SRAM memories are faster but require more space and, therefore, have less usable memory for the same amount of space. [50]

To realise an SRAM cell, transistors in a cross-coupled inverter configuration are generally used. There are different implementation options, which sometimes differ in the number of transistors. One variant of building SRAM cells are so-called four transistor cells (4T) as shown in Figure 2.1. In addition to SRAM cells with four transistors, there are also implementations with six transistors (see Figure 2.2). The two additional transistors are used to control access to the memory cell during read and write operations. This also reduces static power. With only four transistors, a constant current flows through pull-down resistors, increasing the chip's overall power consumption. There are also memory cells with 8 or 10 transistors that utilise additional functions, such as implementing additional ports in a register file, etc., for the SRAM memory. [50]

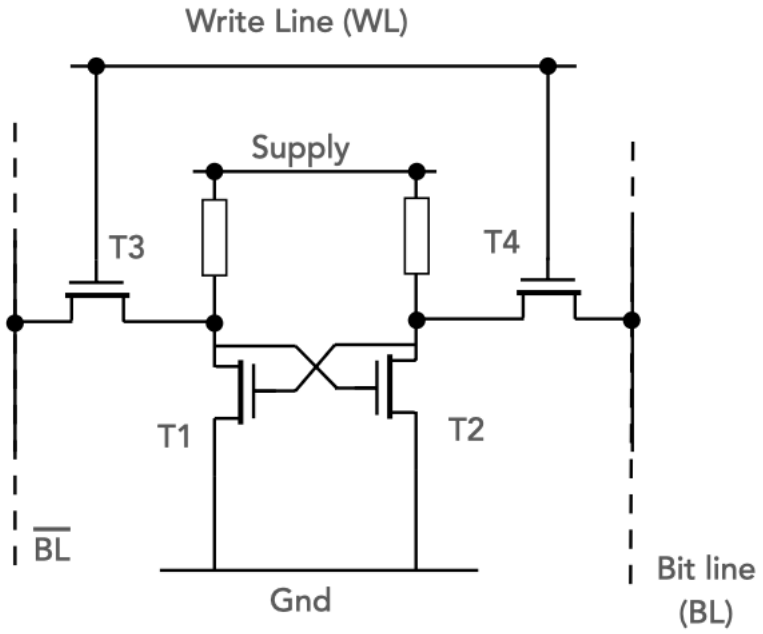

Figure 2.1.: Four transistor version of an SRAM cell [50]

Several cells are arranged in a matrix, and each cell can be addressed individually. Usually, a series of cells is selected, and their contents are read out in sequence. Cells are often equipped with two-bit lines, one of which returns the bit information and the other the inverse. This serves to guarantee integrity. If the values on both lines are the same, there is an error, and the cell has to be read out again. Access to specific cells is activated by the so-called word line. It controls the access control transistors, controlling the connection between the cell and the bit lines. Both lines are used for read and write access. SRAM and DRAM are among the most commonly used memories. SRAM is often used in microcontrollers and microprocessors as it is faster and requires less power in the idle state. It is also easier to control than DRAM, as there is no

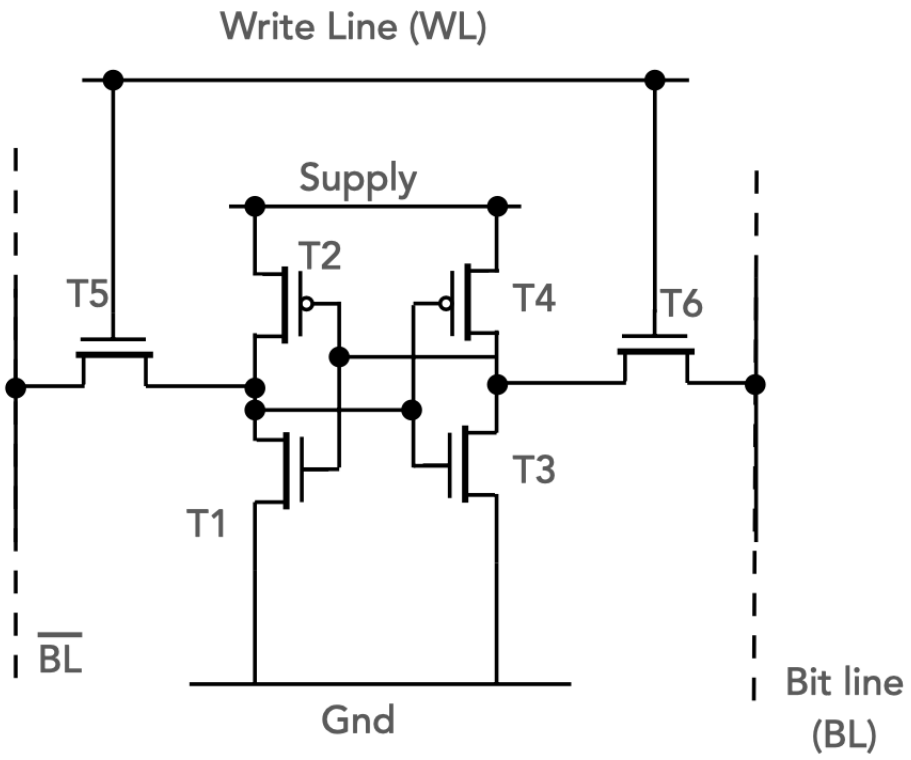

Figure 2.2.: Six transistor version of an SRAM cell [50]

refresh. Although DRAM has a higher storage capacity for the same size, this is not usually required for microcontrollers. Compared to flash memory or Electrically Erasable Programmable Read-Only Memory (EEPROM) technology, SRAM memory also has no limit on the number of read/write cycles, making it perfect for real-time data acquisition applications. [50]

2.4.1. SRAM Based Physical Unclonable Functions

Microscopic variations in the dopant atoms in the channel area of the Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs) cause differences in the threshold voltage of the transistors in an SRAM cell. The transistors that form the cross-coupled inverters are designed to be exceptionally weak, allowing them to switch from 0 to 1 easily during a write operation. For this reason, these transistors are highly susceptible to intrinsic fluctuations at an atomic level, which are not under the control of the manufacturing process and are independent of the position of the transistor on the chip. The read and write process is not affected in regular operation as the SRAM cells are designed with appropriate width/length ratios between the individual transistors. During the power-on process, the cross-coupled inverters in an SRAM cell are not exposed

to any external signal influence. Therefore, due to intrinsic parameter deviations of the transistors, even the slightest voltage differences tend to become either a 0 or a 1. This is due to the amplification effect of the inverters acting on the output of the other inverter. For this reason, the probability is relatively high that the same SRAM cell will start in the same state after being switched off and on again. However, the individual SRAM cells behave independently of each other. [30] A binary string is obtained if the random states of a large number of SRAM cells are read out directly after switching on. This binary character string is unique for each SRAM memory due to production variations at the atomic level and can, therefore, be seen as a kind of digital fingerprint. In contrast to other PUF implementations described in the literature, the binary string can be used directly without going through a quantisation process. This reduces the complexity of the measurement circuit and makes SRAM memory an ideal candidate for PUFs. [30]

The binary strings read from the uninitialised SRAM memory are random and can be used as cryptographic keys after processing (including error correction). As an alternative to storing cryptographic keys in non-volatile memory, PUFs are more resistant to physical attacks because the information disappears whenever the device is powered off. It is also impossible to produce an exact physical copy of the memory due to its atomically unique characteristics (hence physical unclonable function - PUF). [51]

3. Related Work

SRAM PUFs are receiving considerable attention from the research community due to their promising application in embedded systems security and the Internet of Things. They have been researched for over a decade and are already in use in a wide variety of areas such as Aerospace & Defense [1], Automotive Security [52], Medical Equipment [53], Industrial IoT [54], Health, Wearables, Smart grid, Home [55] and more.

SRAM PUFs use atomic level production differences to create a kind of digital fingerprint that cannot be physically copied due to its properties. They offer a unique, secure alternative to traditional cryptographic keys stored on non-volatile memory and could be read by potential attackers with appropriate effort. However, there are some challenges to overcome, such as the instability of individual cells, finding suitable error correction methods, problems due to silicon ageing effects, and secure derivation of cryptographic keys, which are the subject of past and current research.

This chapter presents some of the most relevant papers for this master thesis, including their results. It discusses the general security of SRAM PUFs, reviews the stability of the individual cells under different circumstances, explains selection methods for stable bits in the PUF response and examines how other researchers have proposed a secure authentication method based on the SRAM PUF.

At the beginning of the work for this master thesis, there was primarily research on SRAM PUF measurements and implementations for complex Field Programmable Gate Arrays or external SRAM memories, but little research on the use of SRAM PUF on commodity hardware like the ESP32. However, while writing this thesis, several papers and software were published in this regard. However, most are focused on measuring and analyzing the PUF responses and not on developing an authentication method. One paper [56] deals with both SRAM PUF measurements and analysis and an authentication method for the ESP32; this is discussed in more detail in a subchapter.

3.1. Intrinsic ID

Intrinsic ID is a company that specializes in security Intellectual Property (IP) for embedded systems based on PUF technologies. Their IP can be delivered in software or hardware and applies to virtually any chip. It is used as a hardware root of trust to validate payment systems, ensure secure connectivity, authenticate sensors, and protect sensitive government and military data and systems. Their solutions are used and certified on millions of devices, including EMVco, Visa, CC EAL6+, PSA, IoXT and governments worldwide. [57]

3.1.1. Silicon Ageing

Over time, an Integrated Circuit (IC) changes due to ageing processes. These physical changes usually alter the circuit's functionality, resulting in degradation that can ultimately lead to the failure of the entire IC. The most important degradation effects include Negative Bias Temperature Instability (NBTI), Hot Carrier Injection (HCI), Electromigration and Time-Dependent Dielectric Breakdown (TDDB). Ageing effects like these reduce the reliability of a PUF over time and lead to multiple failures. Intrinsic-ID tested the ageing effects by exposing ICs to high temperatures (80°C) and high voltage for 130 days and checking SRAM PUF values against reference values at room temperature. The noise in the SRAM PUF increased from 5% to almost 15% in this test. [58]

3.1.2. SRAM Aging Mitigation (Anti-Ageing)

Anti-ageing strategies have been developed to counteract the effects of ageing and ensure the reliability of SRAM PUFs without affecting PUF quality, safety, or efficiency. One possible solution for anti-ageing is to write the opposite value of an SRAM cell after power-up. If an SRAM cell adopts the value 0 after power-up, 1 is written to this address and vice versa. Intrinsic-ID experiments showed that the noise in the PUF drops to below 10% almost immediately after applying these anti-ageing procedures and follows a further downward trend over the next 100 days. If such anti-ageing procedures are applied from the start, they can also reduce the natural PUF noise. [58]

3.1.3. Influence of Temperature Variation

Since 2003, Intrinsic-ID has conducted billions of SRAM PUF measurements from manufacturers and production facilities worldwide to measure noise behaviour. These tests were carried out at temperatures between -55°C and +125°C. The results show that at a room temperature of around 25°C, the PUFs exhibited

the lowest noise behaviour, and at a temperature of +125°C, the highest. The diagram in Figure 3.1 shows the sensitivity of the SRAM PUF noise at changing temperatures as part of a typical temperature fluctuation measurement for military applications. [58]

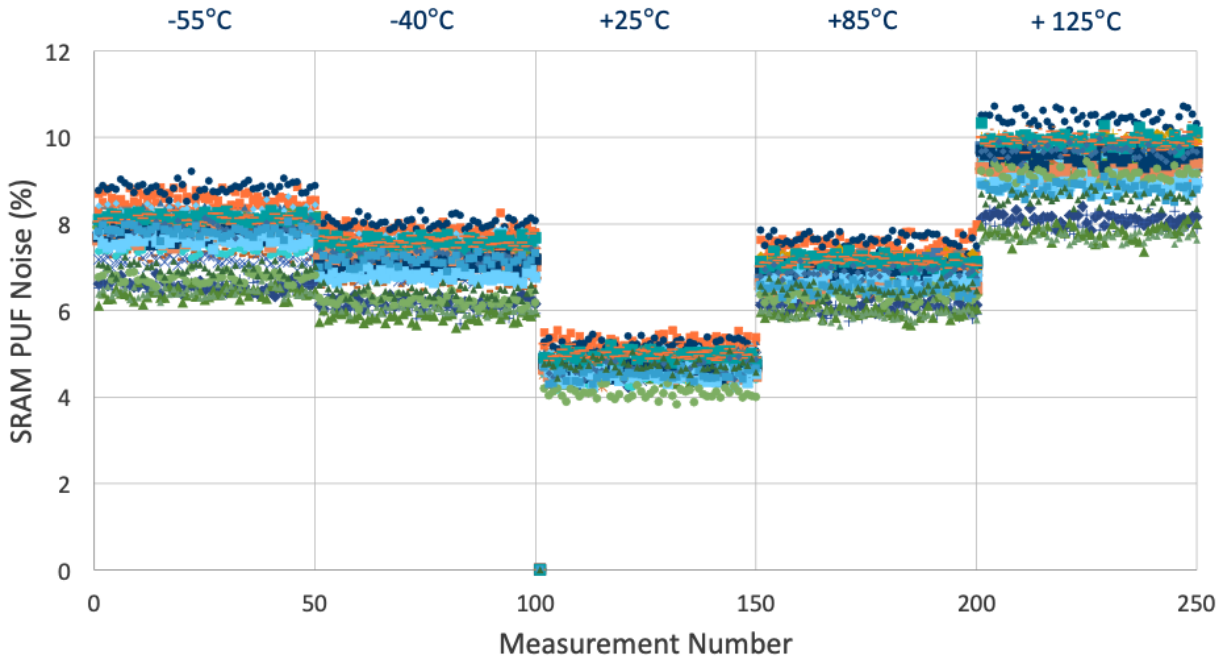

Figure 3.1.: Influence of temperature variation on SRAM PUF noise [58]

The research results of Intrinsic-ID provide a profound insight into the behaviour of SRAM memories. Compared to this work, their long-term measurements were conducted over a significantly longer period and in considerably higher numbers. Due to their vast data set, they can draw more meaningful conclusions and provide more accurate information. However, their research mainly tested commercially available memory modules. In contrast, the integrated Real Time Clock (RTC) slow memory SRAM of the ESP32 was tested as part of this master's thesis, which means that more precise information can be provided for this specific microcontroller.

3.2. Systematic Bit Selection

The authors of "A Systematic Bit Selection Method for Robust SRAM PUFs" [51] proposed a method for selecting reliable SRAM cells for PUF applications. In addition, they investigated the impact of ageing effects, temperature and voltage variations on the stability of SRAM cells.

3.2.1. Ramp Rate Impact on Reliability of SRAM PUF

The ramp rate is the rate at which the SRAM memory is supplied with the supply voltage. Theoretically, the power supply from the Virtual Switching System (ground) (VSS) to the Drain to Drain Voltage (VDD) can be established very quickly, in nanoseconds or less, or very slowly, over several seconds. With a fast ramp rate, particular MOSFETs in the SRAM are neglected; with a fast ramp rate, all MOSFETs are taken into account, resulting in different power-up values in the various cells. The values that an SRAM cell assumes after a power-up can, therefore, sometimes depend on the respective ramp rate. Cells that display inconsistent values at different ramp rates are potentially unstable.

3.2.2. Data Retention Voltage for Strong Cell Selection

In the classic data retention test, the most unstable SRAM cells are identified by writing a 1 to all cells and then reducing the power supply to a critical level. The low current supply is maintained for a while and then increased to the initial value. All cells are then read out. Some unstable cells cannot retain the stored values and change their values to 0. Cells that return a 0 after this test have a very strong tendency towards this value and will most likely also adopt this value during a power-up. The same experiment can be carried out for cells with a strong tendency towards 1 by writing all 0 in the cells and repeating the experiment.
The cells that change their value from 1 to 0 or vice versa at a critically low power supply tend to be strong candidates for PUF, as they are strongly biased and are very likely to take their values at a power-up.

3.2.3. Their Results

The authors of [51] performed some tests to evaluate the power-up stability and reliability. For this purpose, commercially available 64 KBit chips were used, and the power-up time of the SRAMs was controlled using a function generator that allowed the power-up time to vary from 7ns to 100s. However, a consistent power-up time of 7ns was used for the tests. The data retention test described above was carried out in the first test, and the stable bits (which change their value when the power supply is critical) were identified. The state of these cells was then tested after a power-up at different temperatures (25°C, 50°C and 85°C). A ThermoSpot direct contact probe system was used for this purpose. The chips were activated 1000 times at all temperatures, and the states of the cells were measured. It was found that the number of unstable cells increased at high temperatures. The same experiment was then carried out with different supply voltages. Here, it was found that the number of unstable cells decreases when the supply voltage is increased. Finally, the authors tested the ageing effect of the cells. Here, 40% of the SRAMS was initialized with the value 1

and 60% of the SRAM with the value 0. They then read out the start-up values of all SRAM chips 1000 times to check the reliability of the selected cells. It was found that with sufficient supply voltage ($0.58V$), all previously selected stable cells were still stable. Controlled ageing, therefore, did not cause any deterioration in the stability of cells that were already stable beforehand.

As part of their research, a bit selection procedure was presented that can be used to find the stable bits in an SRAM that remain stable even with a change in circuit noise, voltage, temperature changes and ageing. The proposed bit selection procedure offers a promising solution for identifying stable bits. Unfortunately, these results could not be verified within the scope of this master's thesis. The RTC SLOW memory SRAM is embedded, so the direct current supply and the ramp rate cannot be controlled. Nevertheless, the results provide indirect help in finding stable bits. Essentially, external interference factors are deliberately used, and the SRAM values are regularly compared with each other to find stable bits. In this master's thesis, stable cells were extracted based on the long-term measurements, the measurements at different temperatures and the measurements at different voltage supplies.

3.3. Vulnerabilities of Memory-Based PUFs

The security of PUF keys is based on two features: uniqueness and uniformity. Uniqueness states that if at least two devices generate a PUF response, these must be random, and the difference between these two responses should have a Hamming distance of around 50%. Uniformity means that the distribution of 0 and 1 in a PUF response should be uniform. To achieve this, the Hamming weight of the PUF responses should be close to 50%. However, [59] showed that this uniform distribution is not sufficient and opens attack vectors.

[59] addresses several aspects of memory-based PUFs that they claim have been overlooked by research. They identify that the PUF responses of architectural and layout similarities of SRAM chips from the same manufacturer with the same model number have correlations. To save costs, many manufacturers also reduce the size of SRAM memories, making them more susceptible to failure and less robust. Some of the different SRAM cell structures (e.g. 4T, 5T, 6T, 7T, 8T and 9T) sometimes lead to biased results. Although symmetric SRAM architectures are the preferred choice of SRAM PUFs, they can produce deterministic, biased outputs due to asymmetric manufacturing technologies.

Process variations are intrinsic phenomena in the manufacturing process that introduce variability (for example, through transistor attributes, interconnect lines, and dielectric layers). The systematic variations cor-

relate with the chip's layout, the silicon wafer's characteristics, the manufacturing site and the technology. However, the random process variation is entirely indeterministic. Patterns can be learned from systematic process variations through statistical analysis. These patterns significantly influence the start data in the SRAM, which can lead to a bias.
To summarize, architectural similarities in SRAM memory and process variations can compromise the security and uniqueness of memory-based PUFs. These circumstances create deterministic and correlating properties, increasing vulnerability to non-invasive attacks by allowing the deterministic properties to be used as side-channel information.

The authors of [59] also found that the use of error correction techniques commonly used in PUF applications can contribute to lower PUF security. Since SRAM PUFs primarily use noisy data and a certain percentage of SRAM cells are unstable, the unstable values are, in many cases, corrected using error correction techniques. As fewer bits are now relevant for generating a PUF response, attacks are more accessible as they result in a much smaller number of patterns to be tested. This reduces the time and resources required by an attacker.
To ensure the security of a PUF, these should ideally have an even distribution of 0 and 1 across all bit positions. This results in a binomial distribution of responses, which, if implemented correctly, resembles a Gaussian distribution. In practice, however, this is not checked sufficiently, which can lead to distortions. If the standard deviation is not adjusted precisely, this can impair the reliability of the PUF. The authors recommend carefully checking the response distribution's uniformity to ensure that the PUF responses are genuinely random and uniformly distributed.

3.3.1. Mitigations

To ensure the security of PUFs, the authors of [59] recommend the following four measures:

- **Avoiding Error Correction Schemes:** Error correction schemes reduce entropy and increase attack risk. If error correction schemes are omitted, attackers have to guess exact answers and cannot quickly reduce the number of possible patterns. There are ways to make error correction unnecessary through specific SRAM designs and selection methods, increasing security, but these approaches are technically more demanding and only sometimes applicable.
- **Using Properly Designed Fuzzy Extractor:** Fuzzy extractors are often used to correct errors and improve security. The authors recommend using fuzzy extractors that strictly follow the fuzzy min-

entropy boundary condition. Some simple fuzzy extractors convert a fixed-length input into a fixed-length output but do not consider the min-entropy, thus jeopardizing security.

- **Pattern Distribution-Aware Challenge Response Pairs (CRPs):** As mentioned before, the bit patterns should have an even distribution of 0 and 1. The authors recommend selecting SRAM addresses during the login and registration phases to ensure an even distribution of all possible responses.
- **Locality-Aware Challenge Response Pairs (CRPs):** PUF responses generated from specific logical address segments may be more vulnerable than those from other segments. A heuristic attack can be prevented by selecting less vulnerable logical address segments for the generation of CRPs. If an attacker knows the exact layout of an SRAM, this could also pose a risk. It is, therefore, the responsibility of SRAM manufacturers to provide concrete guidelines for the use of PUFs, as only they have precise knowledge of the physical layout of the SRAM.

The results and in particular the mitigations from this paper are essential for this master's thesis. Based on the results, error correction schemes and fuzzy extractors were dispensed with in the authentication system and a bit selection scheme was used instead. Furthermore, the distribution of 0s and 1s in the SRAM was verified, the Hamming weight was selected as the central factor for the security of the SRAM PUF, and the uniqueness between the SRAMs was meticulously inspected by all the microcontrollers tested.

3.4. SRAM PUF on the ESP32 Microcontroller

The paper "ESPuF - Enabling SRAM PUFs on Commodity Hardware" [56] was published in September 2023, during the writing of this thesis. The research topic of the paper shows strong similarities to this master thesis. It addresses the implementation of Static Random-Access Memory Physical Unclonable Function (SRAM PUF) based authentication for ESP32 microcontrollers, the same topic as in this thesis. The researchers conducted a comprehensive literature review, performed measurements on several different microcontrollers of the same type and both confirmed and disconfirmed some assumptions from the literature, developed a proof-of-concept for SRAM PUF-based authentication, and then wrote some security concerns and recommendations for future changes to the microcontrollers used.

The authors particularly opted for the ESP32 microcontroller, as many other researchers primarily use expensive Field Programmable Gate Array (FPGA) boards or similar complex hardware. However, such complex hardware is unsuitable for low-cost deployments and is barely used in real-world applications. ESP32

microcontrollers are inexpensive, have built-in SRAM and are already built into a variety of devices, making them the ideal microcontroller for SRAM PUF authentication use cases.
The authors refer to a statement in [60], in which it is mentioned that after 16 successful startup iterations, individual SRAM cells can already be classified as stable or unstable. However, this statement is not substantiated in the reference paper [60], and the measurement results also refute this statement.

3.4.1. Authentication System

The main goal of their concept is to authenticate an ESP32 microcontroller based on the internal SRAM PUF. Due to a single challenge-response pair (CRP), the SRAM PUF is classified as a *weak PUF* and is directly used to derive a secret key. However, due to the use of Elliptic Curve Cryptography, the *weak* characteristic is removed.
The concept of the paper presumes that the manufacturer already performs PUF measurements on the microcontrollers and creates a PUF challenge. Alternatively, the authors mention that a motivated developer can carry out the measurements, but this process is extensive. The manufacturer or a developer must first create a PUF challenge. The PUF challenge is a bit mask that specifies which specific bits in the SRAM are stable (do not change) and unstable (can change during several power-on processes). An authentication server must also be provided, which derives and stores a shared secret when the PUF challenge is created using the discrete logarithm problem. This shared secret does not need to be stored on the microcontrollers, as it can be generated automatically using the PUF when the microcontroller is powered on without being stored on the device.

They tested a total of seven ESP32s in three different variants:

- AZ-Delivery ESP32 DevKit C V4
- Heltec Wi-Fi LoRa 32 (V1 and V2)
- WT32-eth01

The ESP32 has two integrated SRAM sections, with the first (SRAM1) having a memory of 128KB and the second (SRAM2) having a memory of 200 KB. Only SRAM 1 was used for their tests, and only the first 128 bits were read. They justify this by pointing out that 64-bit SRAM PUF security is sufficient for most applications and that, according to studies, the first 128 bits already have sufficient entropy. The same firmware was flashed onto all test devices for the tests, which underwent 1000 PUF iterations. When analysing the results, the researchers found that across all of the tested microcontrollers, the number of stable bits at 1000

iterations was between 72 and 92, with 128 bits read out.

Their proposed scheme is divided into four different phases. The main participants in these phases are the ESP32 (DuT) and the authentication server. The authentication server aims to authenticate the ESP32 using the SRAM PUF and requires knowledge of the specific PUF challenge for the given ESP32, as the challenge is unique for each ESP32.

First phase

In the first phase, the PUF is extracted from the ESP32 to find the stable and unstable bits. The first 128 bits from the SRAM1 are read out several times, and the stable bits are marked with 1 and the unstable bits with 0. The resulting 128-bit array is referred to as the PUF challenge, which must be known to the authentication server. There is no sensitive data in this bit array; therefore, it can be public. According to their reference papers, stable bits can be distinguished from unstable bits after just 16 measurements. However, the authors proved that even after 16 measurements, some of the previously stable bits exhibited unstable properties. They also devote a separate chapter to this stability in their work. The PUF challenge is stored together with the ID of the respective ESP32, and the PUF response is checked. To prevent the authentication server from gaining possession of the secret PUF response, the authors use elliptic curves (SECP256R1) to check the response without disclosing the raw values of the PUF response to the outside world. To prevent trivial replay attacks, a Diffie-Hellman-like key exchange is implemented. A Raspberry Pi 2 connected serially to the ESP32 was used to extract the PUF challenge in this research phase.

Second phase

In the second phase, the authentication server already possesses the PUF challenge and can, therefore, challenge the ESP32. Elliptic curves are also used in this step to prevent replay attacks.

Third phase

In the third phase, the ESP32 receives the PUF challenge and extracts the PUF response. The PUF response is then reduced to stable bits using the PUF challenge. A random public number is then generated, from which an ephermeral key is generated. A composite point can then be calculated and used as a shared secret

for the respective session. The point can be used as a unique key and an initialisation vector using the binary representation and a Key Derivation Function (KDF). The ID of the ESP32 sent by the authentication server is used as plain text input for the AES Galois Counter Mode (AES-GCM) encryption. The newly created key and the initialisation vector enable symmetric data encryption as part of the AES-GCM encryption, which can be validated directly.

Fourth phase

The fourth and final phase is PUF validation. Once the ESP32 has sent the data, it must be validated by the authentication server. The server must also calculate the shared secret point using the known parameters. Since the ESP32 sends the random number as part of the response, the authentication server has all the data it needs to calculate the encryption key and the initialisation vector and thus decrypt the ciphertext. If the decryption of the ciphertext is successful, the ESP32 is successfully authenticated with the correct credentials.

3.4.2. ESP32 SRAM PUF Evaluation

In addition to designing and implementing the authentication system, the authors also conducted extensive testing with the ESP32 SRAM PUF. It was found that after 1000 iterations, each of the seven boards tested had 72 to 88 stable bits (out of 128), which is sufficient for 64-bit PUF security. The Hamming Distance for each extracted SRAM content was calculated. The results show that the average distance between all boards is 65.5, which corresponds to 51.2%. This confirms the desired random behaviour of the SRAM PUFs. However, 27-bit positions were observed to have the same value between boards from the same manufacturer, confirming results from other papers.

Cell instability was examined as well. It was found that between ten and 20 PUF challenge extractions, an average of four unstable bits can be filtered out. The same number of unstable bits could be filtered out for measurements between 100 and 200. Between 1000 and 2000 measurements, however, there were only two more. This showed that certain cells are more stable than others and flip once every ten power-up cycles, while others only flip once every 1000.

In order to get an overview of the percentage of stable cells and the distribution of unstable cells across all tested microcontrollers, the authors compiled statistics on the stability of the individual cells.

Six different categories were defined to characterise the stability of the individual cells. The stable bits at the top of the table have stayed the same even after more than 2000 PUF challenges, while the unstable bits have already shown unstable behaviour within the first ten extractions. Between stable and unstable, several SRAM cells only became unstable after 100 to 600 extractions (Degree of Instability (DegOI) 2) or after 600 to 2000 extractions (DegOI 1), for example. These were divided into four different Degree of Instability (DegOI) classes, which are shown in Figure 3.2.

Name	Values of deg(x)	Values of ext(x)
Stable	> 2000	$]0; 5 * 10^{-5}[$
DegOI 1	$]600; 2000]$	$]5 * 10^{-4}; 17 * 10^{-4}[$
DegOI 2	$]100; 600]$	$]17 * 10^{-4}; 1 * 10^{-2}[$
DegOI 3	$]50; 100]$	$]1 * 10^{-2}; 2 * 10^{-2}[$
DegOI 4	$[10; 50]$	$]2 * 10^{-2}; 0.1[$
Unstable	< 10	$]0.1; 1[$

Figure 3.2.: Classes of Degree of Instability (DegOI) [56]

The number of bits in each defined class was counted and displayed in a pie chart (see Figure 3.3). This diagram shows that more than half (55%) of the read bits exhibited stable behaviour, and only 11% were completely unstable.

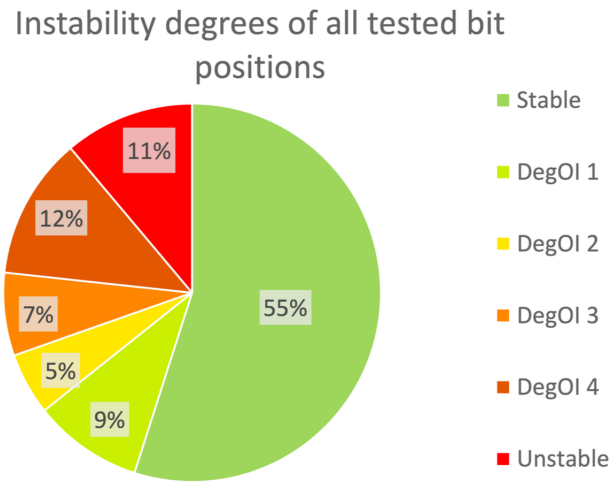

Figure 3.3.: Degree of Instability (DegOI) Distribution [56]

This paper shows the most similarities with this master thesis, as the same microcontrollers were used, the bit stability was evaluated, and an authentication system without an error correction method based on stable bits was developed. Compared to this paper, more than twice as many ESP32s were tested in this master thesis (16 in total from three different series), and the RTC SLOW Memory SRAM was evaluated instead

of the SRAM1. In addition, the entire SRAM memory was evaluated, while in this paper, only the first 128 bits of SRAM1 were tested. Their proposed authentication system is based on elliptic curves and, therefore, needs to be supported by both ESP32 and authentication servers. The system proposed in this master thesis derives API keys and can be used without special server software. The results of the ESP32 SRAM PUF evaluation were confirmed during the evaluations.

4. Methodology

The methodology chapter in this master's thesis explains the systematic approach taken to attain the research objectives. It outlines the literature review conducted to anchor the investigation in the current academic and practical context. It also includes the process of selecting microcontrollers with appropriate SRAM memory. The chapter further elaborates on the methodology followed to perform measurements and analyses, providing insight into the results' empirical basis. Additionally, the chapter presents the research questions of this thesis and describes the development of the authentication system created as a Proof of Concept (PoC).

4.1. Literature Review

First, extensive research on IoT authentication, PUFs in general, and specifically SRAM PUFs was conducted to establish a solid theoretical background. On this basis, several steps were taken, including the procurement, selection and procurement of the required microcontrollers, the development of an API server with a database for storing the measurement data, and the programming of firmware for reading the PUF values and storing them in the database. After several months of testing, comprehensive analyses were conducted based on the data collected in the previous months to gather information on the bit stability and reliability of SRAM PUFs.

Based on the measurement results and literature review, an authentication system was developed and implemented, enabling secure authentication of internet-enabled microcontrollers without storing keys. Different security mechanisms were implemented to ensure the integrity and confidentiality of the transmitted data. All steps were carefully documented and described in the following chapters of this thesis.

4.2. Evaluating Suitable Microcontrollers

One aim of this work is to implement an SRAM PUF-based authentication system. A prerequisite for selecting a suitable microcontroller was that it should be Internet- or network-enabled to authenticate itself to other systems via an IP-based network protocol. It should also have built-in SRAM so that no external hardware is required for the authentication system. Furthermore, it should be commercially available, inexpensive and widely used.
Several microcontrollers with these requirements, such as the ESP32, ESP8266, and RTL8710, were found during a search. Due to the easy and fast availability, as well as the existence of dev boards, the low price, and the high use in a variety of IoT and smart home products, the decision was made in favour of microcontrollers of the ESP32 model.

A total of 16 microcontrollers were procured, of which:

- 7 ESP32-WROOM-32
- 7 ESP32-WROVER-B
- 2 ESP32-S

4.3. Measurements and Analysis

The next step is to check whether the internal SRAM memories are suitable for use as PUFs. The **Technical Reference Manual** [15] of the ESP32 lists multiple SRAM memories: Internal SRAM 1, Internal SRAM 2, RTC FAST Memory and RTC Slow Memory. All options were tested, but the readout of the uninitialized values did not work for all addresses described in the Technical Reference Manual for all SRAM memories. The RTC Slow Memory was the only SRAM that could be read out without any problems and was not yet initialized and not yet written with data during the readout of the entire SRAM directly after starting the controller. It also showed slightly different data for each start-up. Its size of 8KB is also more than sufficient for implementing a PUF. For this reason, the RTC SLOW memory was selected for all further tests, measurements, and the implementation of the authentication system.

Several research questions were defined in order to test whether the RTC SLOW memory is suitable for the use of a PUF:

- How much do the PUF responses change over time (Compared to the initial measurement)?
- How stable are the measurements in general?

- What is the ratio of stable bits to unstable bits?
- Are there enough stable bits to use the data for secure authentication or other cryptographic applications?
- What effects do external influences such as different temperatures, voltages, and time between measurements have on the PUF responses?
- What is the ratio between 1 and 0 in the PUF responses?

Various measurements were carried out to answer these research questions. For this purpose, a server was hosted on the Internet, which has a database and receives and stores measurement data via an API interface. The microcontroller firmware was developed to read the entire RTC SLOW memory (8KB) immediately after the microcontroller is started, cache it, then connect to a WiFi network and lastly, transmit the data to the server via an HTTP POST request. The server stores the measurement data with the tested device's ID and the current timestamp in an SQLite database. In addition to the POST endpoint, the server has a GET endpoint, allowing the measurement data to be viewed or downloaded from a computer. Simple HTTP Basic authentication was implemented as access protection.

As the cells in the SRAM do not change their values as long as they are supplied with sufficient power, the memories have the same data even after a reboot. The power supply for the RTC SLOW memory cannot be controlled by software. In order to measure the changes in the PUF responses, the microcontrollers must be disconnected from the power supply and then rebooted. To automate this process, a commercially available time switch was used, which disconnected several microcontrollers from the power supply at regular intervals and then switched them back on again. This was carried out from 12.04.2022 (`DD.MM.YYYY`) to 13.10.2022 with five microcontrollers and a total of 44,780 measurements recorded.

Subsequently, some manual tests were carried out in which the microcontrollers were manually activated and disconnected from the power supply. The effects of different temperatures, different voltage supplies, and different time intervals (from less than 1 second to several months) between the individual measurements on the PUF responses were examined.

After the data was collected (over 750MB), several analysis scripts were written in Python (with Numpy and Matplotlib) to gain insight into the data. The analysis scripts are very extensive and include:

- The visualization of the entire SRAM (8KB) as a 2D graph

- The visualization of the rate of change between individual PUF responses over a range of several thousand measurements
- Calculation of the Hamming distance between all tested microcontrollers
- Calculation of the Hamming weight over the entire and selected areas in the SRAM
- Measuring and finding the stable and unstable bits, including calculation of their frequency
- and others.

Based on the results of these measurements and the information gathered from the literature research, a concept for an authentication procedure was developed.

4.4. Implementing an Authentication System

Initially, an error correction method based on error correction methods was considered to correct the unstable bits and then derive a key that is always the same. During the literature research, a paper [59] was discovered that mentions that the use of error correction procedures reduces the security of SRAM PUFs, as attackers require fewer opportunities for brute forcing since incorrect attempts can also be corrected. For this reason, a method was chosen in which only stable bits are used to derive a key, meaning an error correction method is no longer necessary. Based on the measurement results, a list with the positions of the stable bits is created for each microcontroller, and a PUF challenge is derived. The PUF challenge is a bitstream of a certain length, where 0 represents an unstable bit and 1 is a stable bit. This PUF challenge is stored in the microcontroller firmware as a hex string. When a microcontroller is switched on, it reads the entire RTC SLOW memory and extracts the stable bits using the PUF challenge. As only stable bits are used, the result of each microcontroller is always the same. To prevent the PUF response from being transmitted directly to the server, Password Based Key Derivation Function 2 (PBKDF2) is used to generate an API token.

A simple authentication server was written to test the authentication, which stores the API tokens of all microcontrollers and can be accessed via the Internet. Hypertext Transport Protocol Secure (HTTPS) with Mutual TLS (mTLS) was configured so the API token is not transmitted in plain text. The microcontrollers thus authenticate themselves via a client certificate stored in the firmware at the network level and then via an API key derived from the SRAM PUF at the application level.

Thus, a secure and reliable authentication system was developed that uses only stable bits, is based on a Physical Unclonable Function and uses a key derivation system that can generate an arbitrary number of keys. Even if an attacker manages to copy the firmware and install it on an identical microcontroller, the attacker cannot authenticate itself because the key is based on the randomness of atomic variations in the manufacturing process of SRAM memories.

5. Setup

This chapter provides a detailed explanation of the thesis's setup. Section 5.1 describes setting up the development environment for programming the firmware and microcontrollers. Section 5.2 explains the process of reading out the uninitialized SRAM and the reasons for choosing RTC SLOW Memory for this master's thesis. Finally, the setup of the measurement server is explained in detail, including the technical details of how the data was stored (see Section 5.3).

5.1. Configuring the Development Environment

To carry out the tests mentioned in the following chapters, the ESP32 under test must be programmed. All purchased ESP32 devboards have a Micro-USB or USB-C interface to program the microcontrollers and to exchange data via a serial interface. For programming the firmware, the PlatformIO Integrated Development Environment (IDE) [61] in Visual Studio Code [62] was used. The installation is straightforward. The desired board and a framework are selected when creating a new project.

All required data will be downloaded automatically, and the `platform.ini` configuration file will open. The desired baud rate can be set here with `monitor_speed = [value]`. This is used to set the transmission speed for serial communication with the microcontroller. Information can only be transmitted if the microcontroller uses the same baud rate as the computer to which it is connected. The required configuration file is relatively short and is shown in Listing 1.

SRAM	Size	Information	Addresses
Internal SRAM 0	192 KB	The first 64 KB can be configured to cache external memory access.	`0x4007_0000` ~`0x4009_FFFF` (instruction bus)
Internal SRAM 1	128 KB	Part of this memory can be remapped onto the ROM 0 address space.	`0x3FFE_0000` ~`0x3FFF_FFF` (data bus) and `0x400A_0000` ~`0x400B_FFFF` (instruction bus)
Internal SRAM 2	200 KB		`0x3FFA_E000` ~`0x3FFD_FFFF` (data bus)
RTC FAST Memory	8 KB	Can be read and written by PRO_CPU and not accessed by the APP_CPU.	`0x3FF8_0000` ~`0x3FF8_1FFF` (data bus) and `0x400C_0000` ~`0x400C_1FFF` (instruction bus)
RTC SLOW Memory	8 KB	Can be read and written by either CPU. The address range is shared by both the data bus and the instruction bus.	`0x5000_0000` ~`0x5000_1FFF` (data bus and instruction bus)

Table 5.1.: Internal SRAMs of the ESP32

```
1 [env:upesy_wroom]
2 platform = espressif32
3 board = upesy_wroom
4 framework = arduino
5 monitor_speed = 115200
```

Listing 1: PlatformIO configuration file

The ESP32 can now be connected to the computer and programmed via the Universal Serial Bus (USB).

5.2. Development of a Firmware to Read Uninitialised SRAM PUF Values

According to the Technical Reference Manual (see [15]), the ESP32 has several SRAM memories listed in Table 5.1. One of them is the so-called RTC SLOW memory. This is an 8 KB SRAM, which is mainly used as retention memory or instruction & data memory for the Ultra Low Power (ULP) coprocessor. The ULP coprocessor is an ultra-low-power processor that remains switched on during deep-sleep mode. The ULP can access peripherals, internal sensors, and RTC registers with minimal power consumption, which is very useful in applications where the Central Processing Unit (CPU) needs to be turned on by an external event, a timer or a combination of both. The RTC SLOW memory can be read and written by both the CPU and the ULP. It is supplied with power during deep sleep mode.

Once the addresses of the various SRAM memories were known, a simple firmware was written that reads

the uninitialised values in the SRAM and outputs them via serial communication (USB). At startup, the firmware initialises the serial communication with a baud rate of 115200. It then reserves a buffer of 8192 bytes (8 KB, size of the RTC SLOW memory) for temporary storage and then attempts to access the content of the configured memory address in hexadecimal format and output it via serial communication. If the memory allocation fails, an error message is issued. The source code for reading can be found in Appendix B.

5.2.1. Reading Internal SRAM 0

Internal SRAM 0 is only available via the instruction bus at memory addresses `0x4007_0000` to `0x4009_FFFF`. However, this cannot be read out directly. An error occurs when accessing these memory addresses with the code mentioned above. A **Core 1** register dump (see Listing 2) is written via serial communication, and the ESP32 restarts.

```
1  PC      : 0x400d1378  PS      : 0x00060830  A0      : 0x800d2565
   ↪  A1      : 0x3ffb2260
2  A2      : 0x3ffc1c60  A3      : 0x40070000  A4      : 0x0000000f
   ↪  A5      : 0x00000004
3  A6      : 0x3ffb8188  A7      : 0x80000001  A8      : 0x800d135f
   ↪  A9      : 0x3ffb2240
4  A10     : 0x3ffb2330  A11     : 0x00000035  A12     : 0x00000003
   ↪  A13     : 0x00000003
5  A14     : 0x00000001  A15     : 0x00000078  SAR     : 0x00000018
   ↪  EXCCAUSE: 0x00000003
6  EXCVADDR: 0x40070000  LBEG    : 0x400863dd  LEND    : 0x400863ed
   ↪  LCOUNT  : 0xffffffff
7
8  Backtrace: 0x400d1375:0x3ffb2260 0x400d2562:0x3ffb2290
9
10 ELF file SHA256: 68d57a8bd4fb5933
```

Listing 2: Core 1 register dump during readout

5.2.2. Reading Internal SRAM 1

Internal SRAM 1 has been read beginning with register address `0x3FFE0000`. However, it was found that a large part of the memory had already been written or initialised. Listing 3 shows a truncated example result read out on one of the microcontrollers when reading out SRAM 1. The method used to visualise the following images in this chapter is described in Section 6.2.4.

```
1 [DEBUG] ESP32 started, will read uninitialized memory
2 288F8438D6886D653ECAD142BE5F6B0F00000000000000000000000000 ...
3 [DEBUG] ESP32 successfully read uninitialized memory
```

Listing 3: Debug output of the ESP32 during SRAM readout

Figure 5.1 shows a 2D representation of the first 8 KB of SRAM 1, in which each box represents a single bit. A black box symbolises a logical 1, and a white box represents a logical 0.

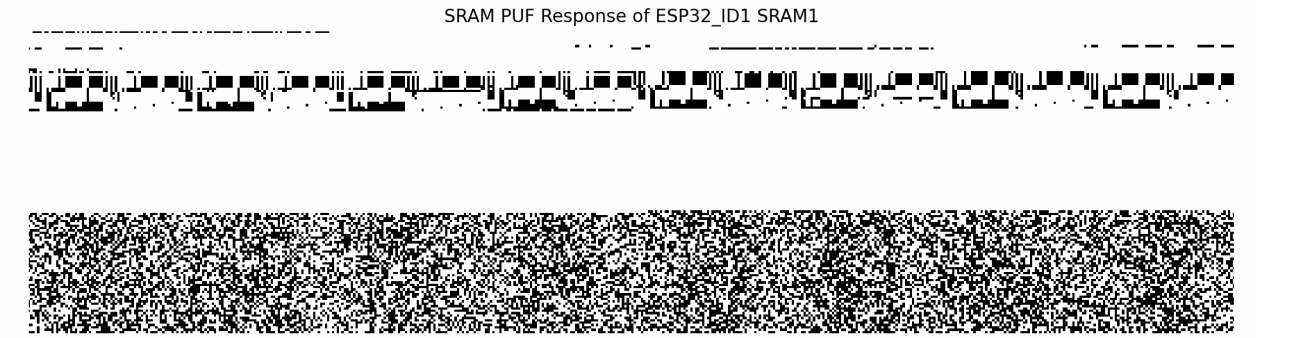

Figure 5.1.: SRAM PUF response of `ESP32_ID1` SRAM1

5.2.3. Reading Internal SRAM 2

Internal SRAM 2 was read starting with register address `0x3FFAE000` and showed similar behaviour to Internal SRAM 1. Some areas have already been initialised or overwritten. Figure 5.2 shows a 2D representation of the first 8 KB of SRAM 2.

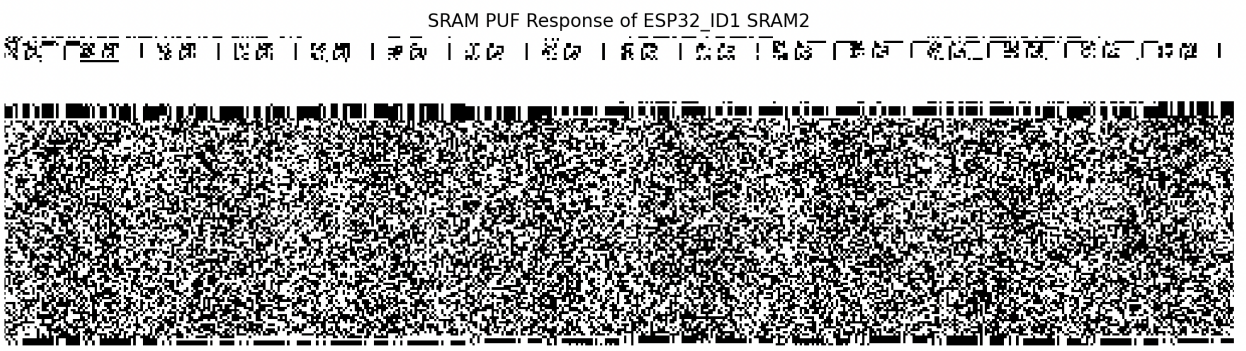

Figure 5.2.: SRAM PUF response of `ESP32_ID1` SRAM2

5.2.4. Reading RTC FAST Memory

Reading out the RTC FAST Memory at address `0x3FF80000` caused the ESP32 to crash, as it did with Internal SRAM 0, and ended in a boot loop. Although the address is on the data bus, the area can only be read and written by the PRO_CPU.

5.2.5. Reading RTC SLOW Memory

The RTC SLOW memory was read out starting with register address `0x50000000` and showed the most promising behaviour of all built-in SRAM memories. There were no problems during reading, and the entire memory of 8 KB appears to be uninitialised and unwritten. In addition, the 0 and 1 ratios seem even and random. A 2D visualisation of the RTC SLOW Memory SRAM is demonstrated in Figure 5.3.

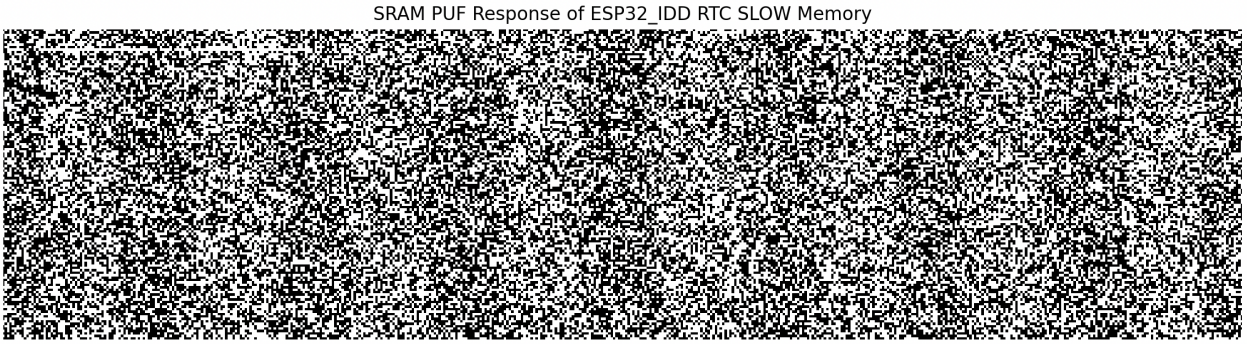

Figure 5.3.: SRAM PUF response of `ESP32_ID1` RTC SLOW Memory

Based on these test results, the decision was made to use the RTC SLOW memory at memory addresses `0x5000_000` to `0x5000_1FFF` for all further experiments and the implementation of the authentication system.

5.3. Measurement Server

Once the selection of a suitable SRAM had been made, the results of the measurements had to be saved. This is the only way to verify later which factors impact the uninitialised SRAM cells and how the readings differ from the first measurement. Thus, a method had to be developed that stores the results of the measurements and can be assigned to the respective microcontrollers. In addition, the time of the measurement must be logged to identify temporal effects on the SRAM PUF values later on.
Various options exist for communicating data from a microcontroller to other systems for storage. For example, microcontrollers can transfer data to other controllers via standards such as Inter-Integrated Circuit (I2C) or Serial Peripheral Interface (SPI), which subsequently store the data. They can also transfer information via a serial interface such as USB, as in the case of debugging. However, since the ESP32 has a built-in Bluetooth and WiFi interface and is mainly used in networks, the decision to use these interfaces to transmit measurements is obvious. For this reason, a simple API server has been programmed in NodeJS (Express Framework) that accepts data and stores it in an SQLite database. The entire code of the measurement server is provided in Appendix B [`index.js`].

5.3.1. Network

The server was rented as a vServer from a Viennese hosting provider. The operating system was Ubuntu Server. The (sub-)domain `sram.christianlepuschitz.at` was set up, and a Domain Name System (DNS) A record was created for the server's IP address. The NodeJS application was accessible on port 3000, to which the ESP32s connected via HTTP after they had connected to a local WiFi network.

5.3.2. Authentication

A simple HTTP Basic authentication was implemented for all API endpoints to protect the server and prevent unauthorised persons from sending data to or downloading data from the server. For simplicity, only a single user was configured, which was used both for uploading the data by the ESP32 and for downloading the collected data.

5.3.3. Database

A simple SQLite database with a single table and six fields was used to store the data of the uninitialised SRAM memories of all microcontrollers over a long period of time. Table 5.2 overviews the fields used, including data type and purpose.

Field	Type	Description
id	`INTEGER PRIMARY KEY AUTOINCREMENT`	Primary key of the table to ensure integrity
timestamp	`DATETIME DEFAULT CURRENT_TIMESTAMP`	Stores the time of the measurement
espid	`INTEGER`	Unique ID to identify single ESP32s
inttemp	`REAL`	Internal temperature of the chip
exttemp	`REAL`	Ambient temperature
data	`TEXT`	The entire content of the RTC SLOW Memory SRAM (8 KB) formatted as a hex string

Table 5.2.: SQLite database fields for storing data

5.3.4. Endpoints

Only two endpoints were implemented in the NodeJS server — one for uploading the data and one for downloading the entire database.

POST `/api/sram`

This endpoint accepts a JavaScript Object Notation (JSON) payload with the following values:

- `espid`
- `data`
- `inttemp`
- `exttemp`

The values `inttemp` and `exttemp` were intended to track the effects of ambient temperatures on the quality of the SRAM PUF but were not used in the measurements.
As soon as data is received at this endpoint, the JSON body is checked for validity, and the data is then written to an SQLite database. The database automatically creates the ID and inserts the current timestamp. This endpoint is used exclusively by the ESP32 during the measurements.

GET `/api/srams`

This endpoint uses a `select * from sram` SQL query to retrieve all values from the database, convert them to JSON, and send them back to the client. As the database was already several hundred megabytes in size after several thousand measurements, this endpoint was only used initially for debugging purposes. The database file was then downloaded from the server and read out directly for the data evaluations at the end of the long-term measurements.

5.3.5. Deployment

During the initial development phase, the NodeJS server was used as a test server to ensure the code's functionality. Subsequently, the server was hosted on a vServer for long-term measurements and could be accessed through a public IP address. The manual tests, which included variable temperature and voltage and the calculation of Hamming distances between all microcontrollers, were operated solely on local networks and were only accessible via an internal IP address. To simplify the server setup process, a Docker container was created, and a simple Docker Compose file was written. This facilitated quick installation on all servers, both on the Internet and locally, as only one container had to be created and the corresponding port released.

6. Measuring and Analysing the SRAM

This chapter focuses on the practical implementation of the first part of this work. It explains how measurements were taken over several months, including tests at different ambient temperatures and with varying voltage supplies. Finally, it analyzes and evaluates all measurements, which serve as the basis for the second part of this thesis; the development of an authentication system based on SRAM-PUF, which is covered in detail in the following chapter.

6.1. SRAM PUF Measurements

Once the measurement server was set up and the code for reading the uninitialised SRAM values was in place, the microcontrollers needed to send the data to the measurement server regularly. To achieve this, the firmware had to be extended to connect the microcontroller to a WiFi network upon start-up, read out the uninitialised SRAM, convert it to JSON, and send it to the measurement server via POST request. Additionally, since the RTC's SLOW memory is always powered, a method had to be devised to disconnect and reconnect the microcontroller from the power supply periodically.

To send the read SRAM values to the measurement server, the ESP32 must first connect to Wi-Fi. The `WiFi.h` library must be integrated, and the Service Set IDentifier (SSID) and Wi-Fi password must be configured to do this. A connection to the WiFi network can then be established using `WiFi.begin()`. When a network connection is established, the `WiFi.status()` function returns the value $true$. The local IP address can be read out using `WiFi.localIP()`. A minimal example of establishing a connection to a Wi-Fi network is illustrated in Listing 4. If the network connection is successful, the serial output appears as in Listing 5.

```
#include <WiFi.h>

const char* ssid = "[SSID]";
const char* password = "[PASSWORD]";

void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());
}

void loop() {

}
```

Listing 4: Connecting an ESP32 to a WiFi network [63]

```
Connecting
.
Connected to WiFi network with IP Address: 192.168.178.66
```

Listing 5: Connecting an ESP32 to a WiFi network

Once the ESP32 is connected to the network, it can send data to the measurement server via a POST request. For this purpose, the `HTTPClient.h` library was used. Firstly, HTTP headers are configured using the `http.addHeader("Type", "Value");` function. Two headers have been configured:

- `Content-Type: application/json`
- `Authorization: Basic aSBsaWtlIGN1cmlvdXMgcGVvcGxlIDop`

Next, the JSON body was created. The `espid` was manually configured for each ESP32, while `exttemp` and `inttemp` were set to 500. The SRAM values were incorporated into the JSON string using simple string concatenation. The code for preparing and sending the data is shown in Listing 6.

```
1  WiFiClient client;
2  HTTPClient http;
3
4  http.begin(client,
   ↪  "http://sram.christianlepuschitz.at:3000/api/sram");
5
6  http.addHeader("Content-Type", "application/json");
7  http.addHeader("Authorization", "Basic
   ↪  aSBsaWtlIGN1cmlvdXMgcGVvcGxlIDop=");
8  String jsondata = "{\"espid\":2,\"data\":\"" + String(memory) +
   ↪  "\",\"exttemp\":500,\"inttemp\":500}";
9
10 int httpResponseCode = http.POST(jsondata);
11
12 Serial.print("HTTP Response code: ");
13 Serial.println(httpResponseCode);
```

Listing 6: Connecting an ESP32 to a WiFi network

The entire code for reading the SRAM, establishing the network connection and sending the data to the server can be found in Appendix B [`read_sram_and_upload_to_measurementserver.cpp`].

6.1.1. Long-Term Measurements

The ESP32 has a system API that allows for a restart to be initiated. However, it was discovered during testing that the SRAM values remained the same as the previous measurements after a restart. This is because the SRAM memories continued to receive power during the reboot. As long as an SRAM cell has power, it will retain its status. Since the software cannot interrupt the power supply to the SRAM memories, the entire microcontroller has to be disconnected from the power supply. Various methods were tested to automate this, but in the end, a simple commercial timer was used. Multiple microcontrollers were alternately supplied with power and disconnected from the power supply every 15 minutes. Because the microcontrollers start automatically upon receiving power, the test could be easily automated for an extended period of time. The process was regularly checked via the GET endpoint of the measurement server.

6.1.2. Power Measurements

Several manual tests were conducted to determine whether the power supply affects the stability of the cells. The tests involved connecting the ESP32s to a laboratory power supply and operating them with different voltages. The ESP32 has $3.3V$ and $5V$ inputs, which were used for the measurements. However, the SRAM was not directly supplied with the specified voltage. Instead, only the 3V3 ($3.3V$) or V5 ($5V$) pins on the development boards were used, which is similar to using a battery for power supply. It's important to note that the ESP32 doesn't function below $2.8V$ on the 3V3 pin and below $4V$ on the V5 pin.

The following power supplies were used during the tests:

- $2.8V$ (3V3 pin)
- $3.3V$ (3V3 pin)
- $5V$ (V5 pin)
- $6V$ (V5 pin)
- $7V$ (V5 pin)

Similar to the long-term measurements, the results were sent to the measurement server for storage using an HTTP POST request. Additional variables were sent as identifiers in the JSON to distinguish the measurements from the long-term ones. The evaluation results are explained in Section 6.2.5.

6.1.3. Overwriting the Start-Up Values

An experiment was conducted to check if overwriting the SRAM affected the stability of the bits. The SRAM was read and overwritten directly, and the ESP32 was restarted. The resulting data was sent to the measurement server and stored via HTTP post request, similar to the long-term measurements. Additional variables were included in the JSON to distinguish these measurements from the long-term measurements.

6.1.4. Testing the Effects of Temperature

To test the impact of ambient temperature on bit stability, measurements were conducted at room temperature and about -20°C. After one hour of placing the microcontrollers in a freezer, 20 measurements were taken at short intervals and sent to the measurement server via HTTP post request. Additional variables were sent as identifiers in the JSON to differentiate the measurements. While attempting to include the chip's exact temperature in the results, the temperature sensor in the ESP32 was accessed, but the values were too inconsistent and inaccurate for use in evaluations. Further details on the evaluation results can be found in chapter Section 6.2.6.

6.2. Analysing and interpreting the data

During a 6-month period, a total of 44,778 measurements were recorded and saved in the measurement server database. To assess the data, the database was downloaded and analysed. An SQL query (refer to Listing 7) was used to determine the number of measurements per `ESPID` in the database, providing an overview of how many measurements were performed on each microcontroller. This overview is displayed in Table 6.1.

```
1 SELECT espid, COUNT(*) AS entries FROM sram GROUP BY espid;
```

Listing 7: SQL query to count measurements per microcontroller

ESP32 ID	ESP32 Series	Measurements taken
1	**ESP32-WROOM-32**	9041
3	**ESP32-WROOM-32**	8641
4	**ESP32-WROOM-32**	8896
6	**ESP32-WROOM-32**	8933
12	**ESP32-WROVER-B**	9267
Total		44778

Table 6.1.: Amount of measurements taken per ESP32

Manual measurements were conducted alongside long-term measurements using 16 ESP32s from three different series; ESP32-WROOM-32, ESP32-WROVER-B, and ESP32-S. Table 6.2 presents an overview of all tested microcontrollers, including their Identifiers (IDs), series, and the first 256 bits of the uninitialised RTC SLOW memory SRAM in hexadecimal representation. Unfortunately, the microcontroller with ID 10 is missing due to a fault. Therefore, IDs in all further graphics and tables range from 1-17.

ID	ESP32 Series	First 256 bits of SRAM PUF
1	ESP32-WROOM-32	0xAB3282E4125FD79C1B6118801056B519B672826CD92EE3CA3F872D95B9E8338F
2	ESP32-WROOM-32	0x399a5e40bf6f16a451fd03bd900b4088c00800870ccc162e47a428652277850f
3	ESP32-WROOM-32	0xB6217FD09134D6E068848D6EF2FDE5974D4A4DA379D773D45E7C8C22F380C232
4	ESP32-WROOM-32	0xBB2D28EC14DB0C1DE01670D80ACA0C97C4D0315202DA56F15BF818574AE86592
5	ESP32-WROOM-32	0x83E9F6EE9281136B99414114049113D399BEB53EC300F8D5EAA28857C6846470
6	ESP32-WROOM-32	0xA35217C8D5BC36ECFF7224D24470AA477F8AA5DADC406A3E3C41C013549EE4E1
7	ESP32-WROOM-32	0xE57DA4415100F5F7FE5A0961200985FF0F8308CE006BB93DE5D242C60200FB53
8	ESP32-S	0x5A755E01ABC2A80A3929D12D2DA30207A3EE286662D0D1A98B0D74E5C818E1B6
9	ESP32-S	0xE86A4A26BF0316AEDDD7A6C1003EBF99777C986D54B4DBB8BE5DB82B2AD474D5
11	ESP32-WROVER-B	0xEEA61EF589501BFEF3C53549334FE87FDAF7A32E1F2BB26362301CB811E13629
12	ESP32-WROVER-B	0x399A1E60BB6F17A450FD03BDD00B40A0C108008744ED120E46A428E0627D8507
13	ESP32-WROVER-B	0xF079B60E2241C49438C58CA707BA77F0C45F20320919F097E30F4EF70DCC603C
14	ESP32-WROVER-B	0x966D0244E0F36DCF0F2A02E99326FB16178A115440FC7A3B85666300A3DE4EE8
15	ESP32-WROVER-B	0x3E338C24801174D876D40AEA61727127AB9A456803C27D75DA7EF8949E360540
16	ESP32-WROVER-B	0x1EE974460115A8975E551DEBECA3CF2117D10418D3433E1E537D2C0474380F9F
17	ESP32-WROVER-B	0x0622AFAB516202CFC3E2F7040C481B60A73020B03225B96D48C1E85CBFD2367B

Table 6.2.: First 256-bit SRAM PUF responses of ESP32 devices

6.2.1. Hamming Weight

The Hamming weight measures the number of bits that equals 1. For an SRAM PUF to be considered secure, the Hamming weight should be 50% [59]. To determine whether the SRAM PUF values of the ESP32 meet this requirement, one measurement was taken with all existing microcontrollers under the same conditions. "Conditions" refers to the standard operating voltage for the microcontroller at room temperature. Subsequently, a Python script was written to read the files with the HEX values of the measurements, convert them into a binary string, and output the number of 0s, 1s, and the ratio as a percentage. The percentage indicates the number of bits with a value of 1. This evaluation was carried out for all available microcontrollers, and the results were summarised in Table 6.3 (the percentages are rounded to 2 decimal places). The evaluation shows that the Hamming weight is nearly 50% for all measured microcontrollers. Therefore, the uninitialised SRAM values exhibit a uniformity that positively affects the security of the PUF.

ID	ESP32 Series	Amount of 1s	Amount of 0s	Hamming Weight in %
1	**ESP32-WROOM-32**	32704	32832	49,90
2	**ESP32-WROOM-32**	32711	32825	49,91
3	**ESP32-WROOM-32**	32786	32750	50,03
4	**ESP32-WROOM-32**	32649	32887	49,82
5	**ESP32-WROOM-32**	32484	33052	49,57
6	**ESP32-WROOM-32**	32649	32887	49,82
7	**ESP32-WROOM-32**	32713	32823	49,92
8	**ESP32-S**	32727	32809	49,94
9	**ESP32-S**	32625	32911	49,78
11	**ESP32-WROVER-B**	32877	32659	50,17
12	**ESP32-WROVER-B**	32666	32870	49,84
13	**ESP32-WROVER-B**	32858	32678	50,14
14	**ESP32-WROVER-B**	32647	32889	49,82
15	**ESP32-WROVER-B**	32700	32836	49,90
16	**ESP32-WROVER-B**	32799	32737	50,05
17	**ESP32-WROVER-B**	32638	32898	49,80

Table 6.3.: Hamming Weight per tested ESP32

6.2.2. Normalised Hamming Distance and Hamming Similarity

The Hamming distance is a well-known measure that quantifies the dissimilarity between two character strings of equal length. The Hamming distance between two strings is defined as the number of positions in which the corresponding symbols differ. For instance, given two character strings 10110101 and 10111100, the Hamming distance would be 2, as both strings differ from each other in two positions.
The Hamming similarity measures the similarity between two character strings, denoted and calculated using the following formula ((x, y) represents the two compared character strings):

$$HammingSimilarity(x, y) = 1 - \frac{HammingDistance(x, y)}{n}$$

For a total of 8 characters with two different symbols, the Hamming similarity of the example above would be 75%.

In the analysis context, the focus is on the difference between two character strings rather than their similarity. Therefore, it is not specified by what percentage they are similar but by what percentage they differ. This value is referred to as the Normalised Hamming Distance and is calculated using the following formula:

$$NormalisedHammingDistance(x, y) = \frac{HammingDistance(x, y)}{n}$$

In the following chapters, Normalised Hamming Distance (in percent) is always referred to as Hamming Distance and is used synonymously.

Computing the Normalized Hamming Distance Across All Microcontrollers

A PUF needs to possess unique security features. Each PUF should be unique and distinct from each other. To determine the differences in the PUFs between all the tested ESP32s, each of the 16 devices was measured. The results of the measurements were saved in a file. A Python script (see Appendix B [`hammingdistance_table.py`]) was then used to convert the data into binary representation and calculate the normalised Hamming distance between all the measurements. A normalised Hamming distance of 0% indicates an exact match, while a distance of 100% indicates an inverted match. The ideal normalised Hamming distance, representing the maximum possible difference between two measurements, is 50%. The script outputs the calculated Hamming distances in a table, which is shown in Table 6.4. The table demonstrates how much the measured PUF value differs from the measured PUF values of all other microcontrollers tested. The Hamming distance is consistently close to 50%, indicating that each PUF of the tested devices is unique.

	ESP32-WROOM-32 [1]	ESP32-WROOM-32 [2]	ESP32-WROOM-32 [3]	ESP32-WROOM-32 [4]	ESP32-WROOM-32 [5]	ESP32-WROOM-32 [6]	ESP32-WROOM-32 [7]	ESP32-S [8]	ESP32-S [9]	ESP32-WROOVER-B [11]	ESP32-WROOVER-B [12]	ESP32-WROOVER-B [13]	ESP32-WROOVER-B [14]	ESP32-WROOVER-B [15]	ESP32-WROOVER-B [16]	ESP32-WROOVER-B [17]
ESP32-WROOM-32 [1]	-	50.2	49.78	49.67	49.37	49.79	49.22	49.74	48.96	49.49	49.92	49.9	49.48	49.48	49.28	49.79
ESP32-WROOM-32 [2]	50.2	-	49.98	49.71	50.13	50.11	50.05	49.86	49.62	49.99	13.47	49.73	49.9	49.97	49.87	50.11
ESP32-WROOM-32 [3]	49.78	49.98	-	50.2	49.8	50.11	50.07	49.67	50.4	50.33	49.76	49.93	49.79	50.36	50.31	50.06
ESP32-WROOM-32 [4]	49.67	49.71	50.2	-	49.32	49.47	48.88	49.75	48.99	49.55	49.6	50.1	49.57	48.98	49.21	49.42
ESP32-WROOM-32 [5]	49.37	50.13	49.8	49.32	-	49.92	49.12	49.82	49.56	49.61	50.01	50.0	49.51	49.72	49.25	49.84
ESP32-WROOM-32 [6]	49.79	50.11	50.11	49.47	49.92	-	49.17	49.66	49.39	50.06	49.84	49.98	49.68	49.06	49.47	49.9
ESP32-WROOM-32 [7]	49.22	50.05	50.07	48.88	49.12	49.17	-	49.72	48.29	48.68	49.77	49.62	48.93	48.52	48.34	49.24
ESP32-S [8]	49.74	49.86	49.67	49.75	49.82	49.66	49.72	-	49.19	49.33	49.61	50.24	49.84	49.42	49.45	49.75
ESP32-S [9]	48.96	49.62	50.4	48.99	49.56	49.39	48.29	49.19	-	48.91	49.54	49.67	49.1	48.71	48.29	49.52
ESP32-WROVER-B [11]	49.49	49.99	50.33	49.55	49.61	50.06	48.68	49.33	48.91	-	49.95	49.82	49.09	49.34	48.99	49.64
ESP32-WROVER-B [12]	49.92	13.47	49.76	49.6	50.01	49.84	49.77	49.61	49.54	49.95	-	49.66	49.67	49.99	49.86	50.0
ESP32-WROVER-B [13]	49.9	49.73	49.93	50.1	50.0	49.98	49.62	50.24	49.67	49.82	49.66	-	49.55	49.98	49.86	49.59
ESP32-WROVER-B [14]	49.48	49.9	49.79	49.57	49.51	49.68	48.93	49.84	49.1	49.09	49.67	49.55	-	49.39	49.09	50.13
ESP32-WROVER-B [15]	49.48	49.97	50.36	48.98	49.72	49.06	48.52	49.42	48.71	49.34	49.99	49.98	49.39	-	48.55	49.7
ESP32-WROVER-B [16]	49.28	49.87	50.31	49.21	49.25	49.47	48.34	49.45	48.29	48.99	49.86	49.86	49.09	48.55	-	49.34
ESP32-WROVER-B [17]	49.79	50.11	50.06	49.42	49.84	49.9	49.24	49.75	49.52	49.64	50.0	49.59	50.13	49.7	49.34	-

Table 6.4.: Hamming distance (in percent) between all tested ESP32

6.2.3. Long-Term Bit Stability Evaluation

As part of the long-term measurements, several thousand tests were conducted on each microcontroller to check the stability of the PUF responses over a long period. The measurement data was used to create text files from the measurements stored in the database. Each text file contained measurements from a single ESP32. A Python script was developed to read all the measurements of a specific ESP32 from the text file and convert them into a binary representation. The first measurement of the microcontroller was saved in a variable, and then the Hamming distance for each subsequent measurement was calculated and saved as a percentage in a list. The list was then visualised as a plot using ‘matplotlib‘. The resulting graph, shown in Figure 6.1 [1] [2], indicates that over several thousand measurements, the PUF measurements gradually diverge from the initial measurement. However, the divergence is only to a minimal extent. The first hundred measurements differ from the initial measurement by between 5.4% and 5.6%. After several months and 9000 measurements, the difference from the initial measurement is around 6.2%. Even the first measurements

[1] Hamming Distance visualisations from first measurement are provided in Appendix A.2.

[2] Hamming Distance visualisations between consecutive measurements are provided in Appendix A.3.

differ from the initial measurement by around 5%. This is approximately the percentage of bits that show the highest instability and change with almost every measurement.

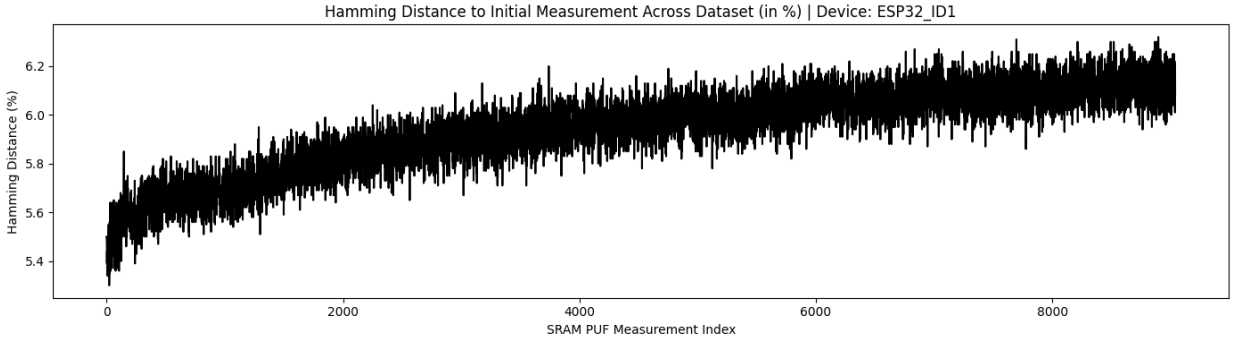

Figure 6.1.: Hamming Distances from first measurement across dataset (`ESP32_ID1`)

Another script (see Appendix B [`hammingweightanddistance.py`] was created to determine the direction in which values move. This script intends to calculate the Hamming weight of each measurement and display it as a plot across all measurements. This aims to determine if more bits are changing from 0 to 1 or the other way around over time. The visualisation result is shown in Figure 6.2 [3]. The graph demonstrates that the Hamming weight, the distribution of 1 and 0, remained consistent across all measurements. As a result, there is no tendency in either direction; the probability that an unstable cell will switch its value to 1 or 0 after numerous power-on cycles remains the same. The blue linear regression across the entire dataset further illustrates this.

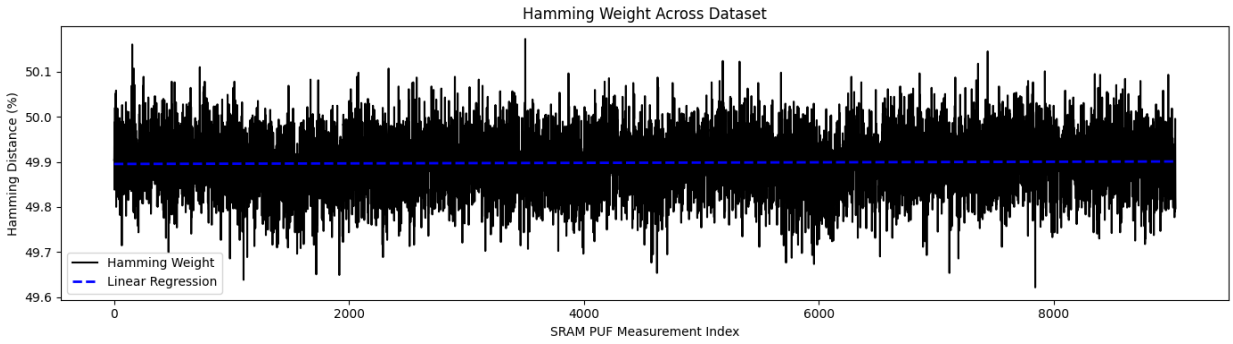

Figure 6.2.: Hamming Weight visualisation across dataset (`ESP32_ID1`)

[3] Hamming Weight visualisations are provided in Appendix A.4.

6.2.4. Visualising SRAM

The database stores the entire content of the SRAM as a Hexadecimal (HEX) string. However, the content needs to be converted into an evaluable form to evaluate the values and answer specific questions. To get an initial understanding of the measured values, another Python script was written. The script reads a hex string, converts it to binary, and creates a 2D graphic. The 2D graph has a size of 512x128 boxes, where each box represents a single bit in the 8 KB SRAM. To achieve this, the first measured value of the `ESP32_ID1` was retrieved from the database using a SQL query (see Listing 8) and saved in a text file.

```
1 SELECT data FROM sram WHERE espid = 1 LIMIT 1;
```

Listing 8: SQL query to get all measurements of a microcontroller per ID

The Python script used to visualise the SRAM reads a file containing a HEX string and converts it to a binary string. This binary string is then transformed into a 2D numpy array with the help of `numpy`. Finally, the output is visualised using `matplotlib`. You can refer to the code for the visualisation in Listing 9. The output is a 2D representation of the uninitialised SRAM, which is 8 KB in size. The visualisation is demonstrated in Figure 6.3.

```python
import argparse
import numpy as np
import matplotlib.pyplot as plt

# Parse command line arguments
parser = argparse.ArgumentParser(description="Visualize bits from a hex string in a 2D graph.")
parser.add_argument("-i", "--input", required=True, help="Input file containing the hex string.")
args = parser.parse_args()

# Read the hex string from the file
with open(args.input, "r") as file:
    hex_string = file.readline().strip()

# Convert the hex string to a binary string
bin_string = bin(int(hex_string, 16))[2:].zfill(8)

# Configuring Pad with zeros if necessary
max_bits = 128 * 512
bin_string = bin_string[:max_bits].ljust(max_bits, '0')

# Convert the binary string to a 2D numpy array
bit_array = np.array([int(bit) for bit in bin_string]).reshape(128, 512)

# Create the visualization
plt.figure(figsize=(12, 3), label="Representation of SRAM Values")
plt.imshow(bit_array, cmap='gray', aspect='auto')
plt.legend()
plt.show()
```

Listing 9: Python script to visualise the SRAM using `numpy` and `matplotlib`

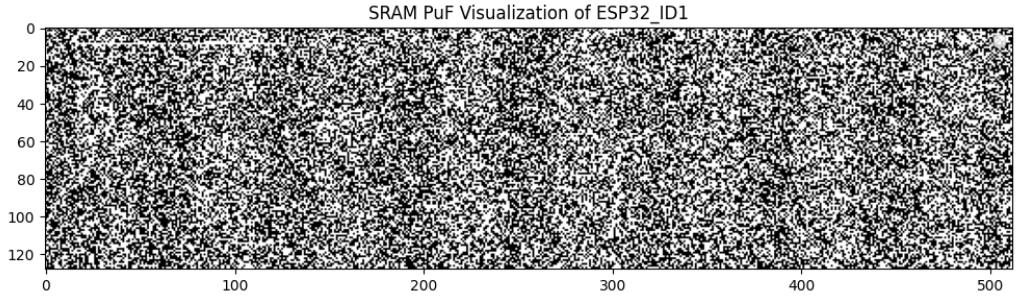

Figure 6.3.: SRAM visualisation (`ESP32_ID1`)

In the provided image, it can be observed that the distribution of 1 and 0 is almost equal, which is also confirmed by the calculation of Hamming weight in the previous chapter. However, upon closer inspection, there is a faint pattern - a horizontal white-black-white-black-white pattern. To further investigate, the script has been updated with the following functions:

- Calculate the Hamming weight per column (from top to bottom) and visualise it as a plot.
- Calculate the moving average for the Hamming weight mentioned above per column.
- Calculate and visualise the linear regression for the whole dataset

The results of the new values are displayed as a subgraph to better observe the correlation with the 2D representation.

These new visualisations were performed for all tested microcontrollers from the three series: **ESP32-WROOM-32**, **ESP32-S**, and **ESP32-WROVER-Bi**. The evaluations show that the black-and-white pattern is independent of the series. In each series, some microcontrollers tend more towards this pattern, while others have a more even distribution of 1 and 0.

Figure A.7 displays the 2D representation of the read-out SRAM of an ESP32 of the **ESP32-WROOM-32** series, along with a subplot that provides more detailed information. The image shows that in certain columns, the probability of the respective cells being 1 or 0 is about 60%, depending on the location.

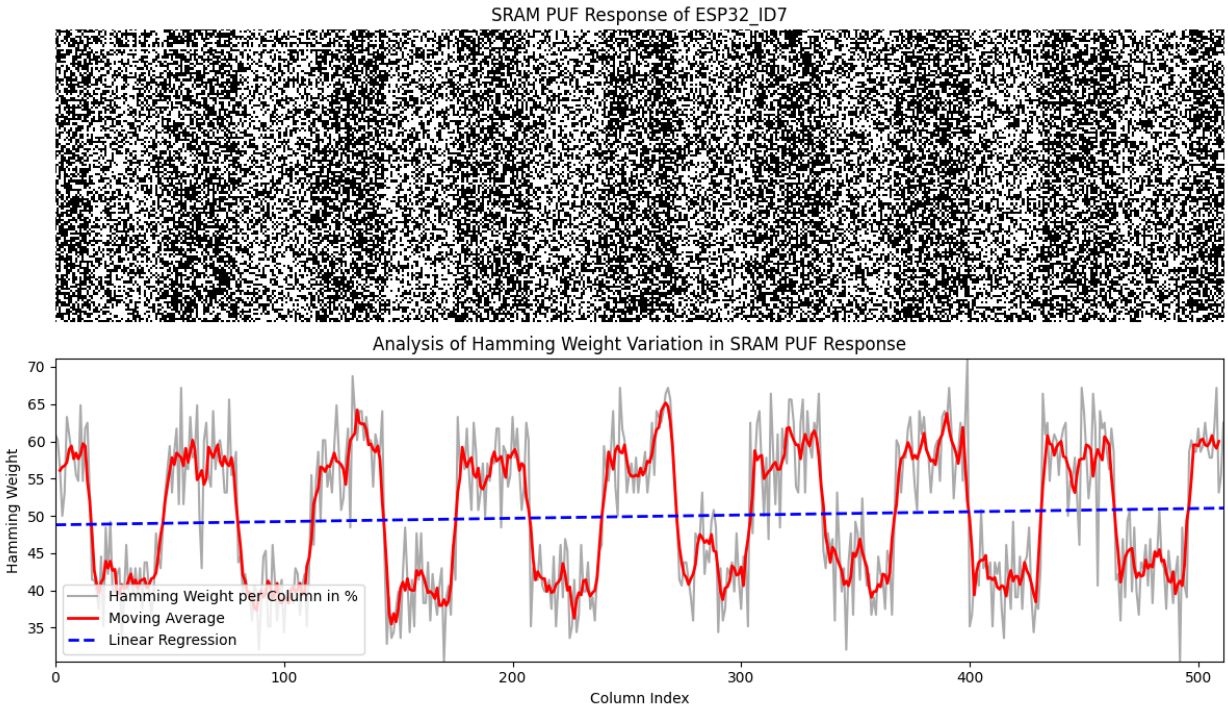

Figure 6.4.: SRAM visualisation with moving average and linear regression (`ESP32_ID7`)

It has been observed that a particular microcontroller from the ESP32-WROVER-B series does not exhibit a pattern in the distribution of 1 or 0 values across its SRAM (see Figure A.11) [4]. This contrasts with other microcontrollers where a clear pattern can be identified. The absence of a pattern in ESP32-WROVER-B indicates a more even distribution of 1 and 0 values across the SRAM, making it a better choice for an SRAM PUF (Physically Unclonable Function) based method. This is because the probability of occurrence of either 1 or 0 at any position in the SRAM is almost 50%, making it highly random.

[4] SRAM visualisations are provided in Appendix A.1.

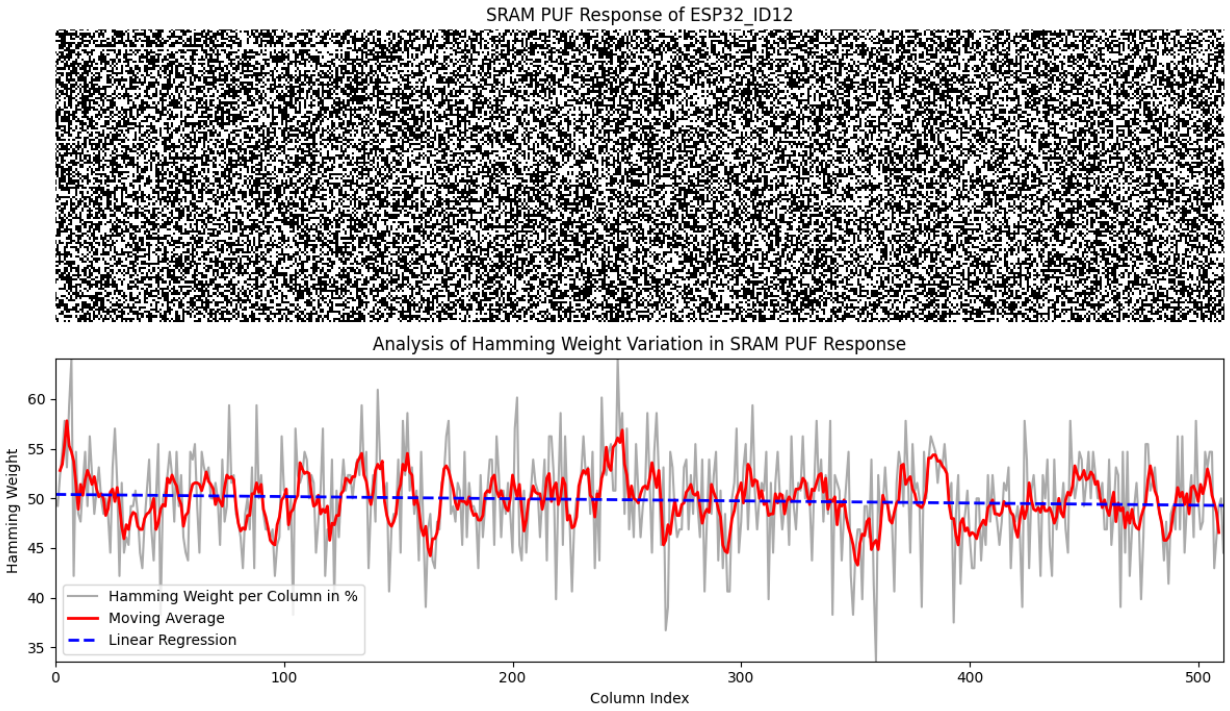

Figure 6.5.: SRAM visualisation with moving average and linear regression (ESP32_ID12)

6.2.5. Examining Variations in Power Supply Voltages

Two ESP32s were tested with different power supply voltages to analyse their effects. The tests were conducted exclusively on the ESP32-WROOM-32 series, as the development boards used to have both a $3.3V$ and a $5V$ input pin, which were used for the tests.

Two microcontrollers, ESP32_ID1 and ESP32_ID7, were used for the tests. ESP32_ID1 was used for long-term measurements and had over 9000 power-up cycles before the experiments. On the other hand, ESP32_ID7 was only used for isolated tests and had less than ten power-up cycles at the time of testing.

In total, sixty measurements were conducted on the two microcontrollers with six different power supplies. Five tests (50 measurements) were carried out via the 3V3 or V5 pins on the development board, and one test (10 measurements) was carried out with power supplied via the micro-USB interface.

The results of the experiments conducted on ESP32_ID1 are presented in Table 6.5, while the results of ESP32_ID7 are shown in Table 6.6 The graphs indicate that the power supply had virtually no effect on the stability of the SRAM cells. The stability remained continuously in the range of 94%, except for an outlier in ESP32_ID1 with a power supply of $7V$ at pin V5, where the stability was only 92.75%. ESP32_ID7 did not exhibit this outlier with the same power supply, even though it was of the same type and manufacturer.

The reason for the minimal effect of varying power supplies on the stability of the SRAM cells may be that the direct power supply of the RTC SLOW memory cannot be controlled and is corrected by the development boards. In order to analyse the effects of different power supplies, several ESP32s were operated with different supply voltages. The tests were carried out exclusively with the **ESP32-WROOM-32** series, as the development boards used have both a $3.3V$ and a $5V$ input pin, which were used for the tests. The microcontrollers used for these tests are `ESP32_ID1` and `ESP32_ID7`. The `ESP32_ID1` was used for the long-term measurements and, therefore, already had over 9000 power-up cycles before these tests. The `ESP32_ID7` was only used for isolated tests and had less than 10 PowerUp cycles at the time of testing. On each of the two microcontrollers tested, 60 measurements were carried out with six different power supplies. Five tests (50 measurements) were carried out via the 3V3 or V5 pins on the development board, and one test (10 measurements) was carried out with a power supply via the micro-USB interface.
The results of the `ESP32_ID1` can be found in Table 6.5, and the results of the `ESP32_ID7` can be found in Table 6.6. The graphs show that the power supply has virtually no effect on the stability of the SRAM cells. The stability is continuously in the range of 94%, whereby there is an outlier in `ESP32_ID1` with a power supply of $7V$ at pin V5 — here, the stability is only 92.75%. The `ESP32_ID7` does not have this outlier with the same power supply, although it is of the same type and manufacturer.

A varying power supply has virtually no effect on the stability of the SRAM cells, which may be due to the fact that the direct power supply of the RTC SLOW memory cannot be controlled and is corrected by the development boards.

ID	Power Supply	Voltage	Tests	Stable (in %)
1	3V3	2,8V	10	94,29
1	3V3	3,3V	10	94,25
1	V5	5V	10	94,14
1	V5	6V	10	94,31
1	V5	7V	10	92,75
1	USB	USB	10	94,48

Table 6.5.: Effects of varying power supply on bit stability (`ESP32_ID1`)

ID	Power Supply	Voltage	Tests	Stable (in %)
7	3V3	2,8V	10	94,36
7	3V3	3,3V	10	94,54
7	V5	5V	10	93,87
7	V5	6V	10	94,54
7	V5	7V	10	94,41
7	USB	5V	10	93,63

Table 6.6.: Effects of varying power supply on bit stability (`ESP32_ID7`)

6.2.6. Examining Ambient Temperature Variations

Two microcontrollers were subjected to a freezing temperature of -20 °C for an hour each to study the impact of ambient temperature on the stability of SRAM cells. Each microcontroller was read twenty times, and the results of the study are presented in Table 6.7. The results indicate that the bit stability of both microcontrollers was around 94% at a temperature of -20 °C.

From the measurements obtained, one reading from each microcontroller was randomly selected and compared to a reference measurement taken at room temperature and using the same power supply. The comparison revealed that the measurements at -20 °C differed 10.07% and 9.75% for `ESP32_ID7` and `ESP32_ID1`, respectively, compared to the reference measurement at room temperature.

The results indicate that while the SRAM PUF demonstrates a stability of around 94% at a constant ambient temperature, the stability drops by a few percentage points as the ambient temperature changes. Therefore, maintaining a constant ambient temperature could increase the reliability of the SRAM PUF.

ID	Power Supply	Voltage	Tests	Stable (in %)
1	USB	5V	20	94,51
7	USB	5V	20	94,47

Table 6.7.: Effects of the ambient temperature on bit stability

7. Implementing an Authentication System

A simple authentication system was designed and developed after conducting research in Chapter 3 and carrying out tests in Chapter 6. This authentication system stands out from others proposed in the literature because it does not require any special key exchange procedures, additional external hardware, error correction procedures or server software. It is particularly suitable for authentication via HTTPS or other protocols that use an API key or similar.

The SRAM PuF is not solely used for authentication but serves as an additional factor. With the presented method, it is possible to carry out several measurements straightforwardly and use them to extract stable bits and generate unique keys for a microcontroller. These keys can be stored on a server system and assigned to a microcontroller. The microcontroller only needs to have a PUF challenge, which is stored locally or read externally (e.g. from a server) after the microcontroller has been started. The PUF challenge does not contain sensitive data and does not need to be stored or transmitted securely.

After this setup process, a microcontroller can read the stable bits in the SRAM directly after the startup process and generate its key using a key derivation procedure to authenticate itself with a server.

7.1. Concept

The authentication system setup is divided into four phases:

1. **Measurements:** The RTC SLOW memory SRAM of the ESP32 is read out using the method described in Section 5.2. The more measurements carried out, the better the results. All measurements are then saved.
2. **Extraction of the stable bits:** The saved measurement results from step 1 are analyzed, and the stable bits are extracted. This creates a PUF challenge, which is a bit array. Each 1 marks a stable bit, while each 0 marks an unstable bit. The stable bits are extracted from any measurement and stored in a new array, creating a purified PUF response. Errors in certain areas of the SRAM when a microcontroller

is started several times is not a problem, as only stable bits are used. Therefore, the cleaned PUF response is always the same whenever the device is switched on.

3. **Key derivation:** The PUF response is used as input for a key derivation function. In this work, PBKDF2 was used to make brute force attacks more difficult and to generate any number of keys by making minor adjustments.
4. **Authentication:** With the help of the first three phases, a PBKDF2 key can be generated during the setup process, which is stored on the server and assigned to a microcontroller. The microcontroller uses the PUF challenge to extract the stable bits after its start and to derive the PBKDF2 key. Then, the microcontroller can establish a secure connection with the server and authenticate itself using the key as an API token.

The derived key (API key) is always the same as long as the PUF response remains the same and the input parameters of PBKDF2 are not changed. To counteract man-in-the-middle attacks, this authentication takes place via an encrypted channel. In this work, mTLS (Mutual Transport Layer Security) was configured, and client certificates were created. The ESP32 first authenticates itself to the server using a client certificate to establish an encrypted connection, in which the client and server authenticate each other. Subsequently, authentication takes place within the mTLS connection using the SRAM PUF API key.

7.2. Implementation

7.2.1. Generating the PUF Challenge and the API Key

To generate the PUF challenges for each microcontroller, data was extracted from the long-term measurements stored in the database. An Structured Query Language (SQL) query was used to create a list of all SRAM PUF measurements for each microcontroller, which was then saved as a simple text file.
A Python script (Appendix B [`pufchallenge.py`]) was developed to read the measurements line by line and convert them into binary format. These binary representations were then compared with each other. Stable bits were saved as 1 and unstable bits as 0 in a new array. This list of stable and unstable bit positions was output by the Python script in hexadecimal format and can be configured on ESP32.
Additionally, the Python script used a salt and 10000 iterations to derive the PBKDF2 key, configured on the server as an API key and assigned to the ESP32. The script takes only one input parameter, `-i`, to specify the input file. Depending on the number of measurements in the input file, it takes a few milliseconds to several seconds to execute. The optional parameter `-v` activates the verbose mode, which displays the

`password to hash` (i.e., the cleaned PUF response) before it is used as input for the PBKDF2 function. It also shows the number of PBKDF2 iterations and the salt used. An example output of the program is shown in Listing 10.

```
# Normal output
$ python3 pufchallenge.py -i espid7.txt
[ESP32]  PUF Challenge:          f3f3dfffefbabffff3fbff7fb6efbeff
[Server] API-Token:              9b6fa081f05bc4b197f4ff7e79f01...

# Verbose output
[Debug]  Password to hash (bin): 11100101110110001000100001101...
[Debug]  Password to hash (hex): E5D8886851D7DFE5C13850263F0F
[Debug]  PBKDF2:                 pbkdf2:10000:43...647a:9b6fa0...
[ESP32]  PUF Challenge:          f3f3dfffefbabffff3fbff7fb6efbeff
[Server] API-Token:              9b6fa081f05bc4b197f4ff7e79f01...
```

Listing 10: Output of the script `pufchallenge.py`

7.2.2. Development and Configuration of the Authentication Server

An Express.js web application framework for Node.js was used to create a simple server application. In the previous step, the API keys generated for each microcontroller were entered statically in the `apiKeys` object as a key value. Each API key generated in the previous chapter was used, along with a simple identifier, such as `ESPID1`.

The server application had only one endpoint, which checked for the availability of an **HTTP Authorization** header with a **Bearer Token**. When an HTTP request with a bearer token was sent to the server, it checked whether the bearer token (API key) was contained in the `apiKeys` object. If the key was present, the server returned the message `{ID} authenticated`. Otherwise, an `Authentication failed` message was sent. The entire source code of the server can be found in Appendix B [`app.js`]

The authentication server was run in a Docker container like the measurement server. The Dockerfile is

provided in Appendix B [Dockerfile]. To create and start the Docker container with the application, the following two commands were executed (see Listing 11). The application then runs locally on port 8080.

To keep things simple, a Cloudflare tunnel was used to configure a domain and HTTPS. The mTLS configuration, including the generation of client certificates, was also carried out via Cloudflare. Cloudflare publicly documents this process [64] and, therefore, this is not explained in more detail in this thesis.

```
1 sudo docker build -t sramauth
2 sudo docker run -d -p 127.0.0.1:8080:8080 sramauth
```

Listing 11: Building and starting the authentication server container

7.2.3. Authenticating the ESP32

Several steps need to be followed to obtain the PBKDF2 key from the SRAM PUF on the ESP32. First, the entire RTC SLOW Memory SRAM is read out. Then, the `extractBitsFromSRAMToString()` function is utilized to extract the stable bits from the read SRAM and return them. To convert the PUF challenge and the read SRAM into a binary representation, two other functions are accessed; `byteToBinaryString()` and `hexStringToBinaryString()`. The code for this process is shown in Listing 12 and Listing 13.

```
// Convert a byte into a hexadecimal form
String byteToBinaryString(byte b) {
    String result = "";
    for (int i = 7; i >= 0; --i) {
        result += (b & (1 << i)) ? "1" : "0";
    }
    return result;
}

// Convert a hex array into a binary representation and store it as
↪ string
String hexStringToBinaryString(const String& hex) {
    String binaryString = "";
    for (char c : hex) {
        int num = c <= '9' ? c - '0' : tolower(c) - 'a' + 10;
        for (int i = 3; i >= 0; --i) {
            binaryString += (num & (1 << i)) ? "1" : "0";
        }
    }
    return binaryString;
}
```

Listing 12: Helper functions for converting the PUF challenge and the SRAM values (Part 1)

```
// Extract stable bits and return the result as string
String extractBitsFromSRAMToString() {
    String binarySRAM = "";
    for (int i = 0; i < 128; i += 8) {
        binarySRAM += byteToBinaryString(ram_buffer[i / 8]);
    }

    String binaryHex = hexStringToBinaryString(pufChallenge);
    String result = "";

    // Make sure that binarySRAM is long enough
    binarySRAM = binarySRAM.substring(0, binaryHex.length());

    for (unsigned int i = 0; i < binaryHex.length(); ++i) {
        if (binaryHex[i] == '1') {
            result += binarySRAM[i];
        }
    }

    return result;
}
```

Listing 13: Helper functions for converting the PUF challenge and the SRAM values (Part 2)

The function `generateKeyUsingPBKDF2()` is used to derive a key. To ensure ESP32 and the Python script generate the same keys, the same parameters used by the Python script in Section 7.2.1 must be used. These parameters include defining a salt, the number of iterations, and the key length. To perform the PBKDF2 function, the libraries `mbedtls/pkcs5.h` and `mbedtls/md.h` are used. An additional input parameter, a password, is required by PBKDF2. For this, the cleaned PUF response from the previous step is utilized. This response is available as a binary string and must be converted to a byte array using `binaryStringToByteArray()` (see Listing 14 and Listing 15).

```
// PBKDF2 Parameters
const unsigned char salt[] = "christianlepuschitz";
const int iterations = 10000;
const int keyLength = 32; // Desired length of the resulting key
unsigned char key[keyLength]; // Array to store the generated key once derived

// Function to convert a binary string to a byte array
void binaryStringToByteArray(const String& binaryString, unsigned char* byteArray, size_t& byteArrayLength) {
    byteArrayLength = 0;
    for (size_t i = 0; i < binaryString.length(); i += 8) {
        unsigned char byte = 0;
        for (size_t j = 0; j < 8 && (i + j) < binaryString.length(); ++j) {
            byte <<= 1;
            if (binaryString[i + j] == '1') {
                byte |= 1;
            }
        }
        byteArray[byteArrayLength++] = byte;
    }
}

// Generate key using PBKDF2
int generateKeyUsingPBKDF2(const String& extractedBits) {
    unsigned char password[1024];
    size_t passwordLength;
```

Listing 14: Generating the PBKDF2 key (Part 1)

```
    binaryStringToByteArray(extractedBits, password,
    ↪ passwordLength);

    mbedtls_md_context_t md_ctx;
    mbedtls_md_init(&md_ctx);
    const mbedtls_md_info_t* info =
    ↪ mbedtls_md_info_from_type(MBEDTLS_MD_SHA256);

    if (info == NULL) {
        Serial.println("[DEBUG]: Error when generating PBKDF2
        ↪ key");
    }

    int ret = mbedtls_md_setup(&md_ctx, info, 1); // 1 for HMAC
    if (ret != 0) {
        Serial.println("[DEBUG]: Error when generating PBKDF2
        ↪ key");
    }

    ret = mbedtls_pkcs5_pbkdf2_hmac(&md_ctx,
            password, passwordLength,
            salt, sizeof(salt) - 1, // -1 to remove the NULL
            ↪ terminator
            iterations, keyLength, key);

    mbedtls_md_free(&md_ctx); // Clean up

    return ret;
}
```

Listing 15: Generating the PBKDF2 key (Part 2)

The function `generateKeyUsingPBKDF2()` is responsible for generating an API token that can be used for server authentication. As this key must be transmitted through an encrypted channel, client-side authentication must be ensured by incorporating the `HTTPClient.h` library. The server's domain, port, and certificates have been saved as constants. The connection to the server is established by using `http.begin()` (see Listing 16).
The whole source can be found in Appendix B [`esp32_authenticate.cpp`].

7.2.4. Testing

Two groups were formed to conduct authentication process tests. Group A consisted of microcontrollers that had undergone long-term measurements, with several thousand measurements taken to extract stable bits. Group B, on the other hand, consisted of unused microcontrollers, and only ten measurements were taken to extract stable bits.
The evaluations revealed that the authentication success rate for Group A was 100%, while test Group B only had a 40% success rate. To increase the authentication success rate, conducting a sufficient number of measurements under varying conditions is advisable.

```
const char* domain = "authsec.xtn.sh";
const char* path = "/";
const int port = 443;

// Root Certificate
const char* cert = \
        "-----BEGIN CERTIFICATE-----\n" \
        // ...
        "-----END CERTIFICATE-----\n";

// Client Certificate
const char* clientcert = \
        "-----BEGIN CERTIFICATE-----\n" \
        // ...
        "-----END CERTIFICATE-----\n";

// Private Key
const char* clientkey = \
        "-----BEGIN CERTIFICATE-----\n" \
        // ...
        "-----END CERTIFICATE-----\n";

void setup() {
// ...
        HTTPClient http;
        http.begin(domain, port, path, cert, clientcert,
         ↪  clientkey);
// ...
}
```

Listing 16: Client certificate authentication on the ESP32

8. Conclusion

This thesis has shown that the ESP32 is a good candidate for Static Random-Access Memory Physical Unclonable Function (SRAM PUF)-based authentication. The Real Time Clock (RTC) slow memory SRAM is particularly suitable as it is entirely uninitialised during readout. 16 ESP32 microcontrollers from three different series were used for testing, and the SRAM PUF values of all tested microcontrollers were found to be completely unique and differed from all others by approximately 50%. Furthermore, the distribution of 0 and 1 is even across all microcontrollers, which has an exceptionally positive effect on the safety of the SRAM PUF. However, it was found that individual models are biased across all series tested, meaning that SRAM cells tend to be 1 or 0 in certain areas. This tendency is up to 65% in some regions of the SRAM.

Over several months, more than 44,000 measurements were performed and evaluated with different ESP32. This analysis aimed to determine the stability of the PUF response over a certain period and to check whether the cells tend towards a 0 or a 1 over time. The results show that measurements carried out in quick succession (on the same day) show a difference of around 5.5%, while this difference increases continuously over time and is as high as 6.2% after six months. Although the values in the uninitialised SRAM change over a long period, the 0 and 1 distributions remain constant at around 50%.

Experiments on bit stability with a variable supply voltage have shown that a supply voltage that is either too high or too low does not affect the stability of the cells. However, this may be because the voltage supply of the RTC SLOW memory was not controlled directly during the tests, but the ESP32 was supplied with power via pins V5 and 3V3 on the development boards. The voltage applied directly to the RTC SLOW memory may have been consistent, which means that the stability of the bits remains consistent with a variable supply voltage.

In addition, tests were conducted to determine the effects of a variable ambient temperature on the stability of the SRAM cells. The results show that the stability at room temperature and at -20°C was around 94%.

However, a difference of around 10% was found when comparing individual measurements from different test groups. The difference between two measurements from the same test group (same temperature) was no more than 5-6%.

It could be shown that 16 measurements are insufficient to identify stable bits (as already confirmed by [59]). The long-term measurements also showed that silicon ageing (as mentioned in [58]) impacts SRAM cells and that PUF responses change over time.
An authentication system that utilises the submicroscopic variations in the manufacturing process of SRAM memories as a kind of fingerprint to authenticate microcontrollers via IP-based protocols was developed based on the results from the tests and the findings from the literature research. The proposed authentication system distinguishes itself from other systems proposed in the literature by its simplicity, the absence of error correction methods and the fact that there is no requirement for any specialised server software to enable seamless integration into existing authentication systems. A Proof of Concept (PoC) was developed to identify stable bits in the SRAM based on collected SRAM PUF measurements and to derive a PUF Challenge. This PUF challenge extracts the stable bits in the SRAM and derives an authentication key based on a key derivation function (KDF). Only the ESP32 for which the PUF challenge was generated can derive the required authentication key and thus authenticate itself. Even if an attacker succeeds in cloning the firmware from an ESP32 and installing it on a large number of identical devices, none of them will be able to authenticate themselves, as the device fingerprint differs from the original device in terms of submicroscopic manufacturing variations in the SRAM.

In the PoC, this authentication key is utilised as an additional factor to an underlying mTLS authentication to prevent the key from being compromised by an Man-in-the-Middle (MITM) attack.
In summary, detailed measurements and analyses were performed, and all of the previously defined research questions were answered. In addition, a PoC for an SRAM PUF-based authentication system was developed, which showed promising results and does not require specialised server software. This system can also be extended to other encrypted network protocols with minor modifications.

8.1. Future Work

Based on the findings in this thesis, the following future research topics were identified:

- Examination of the minimum number of measurements required to achieve specific authentication

success rates.

- Investigation of possible security vulnerabilities and attack vectors in the proposed authentication system.
- Extension of the authentication system to generate multiple PUF challenges per microcontroller and implement key rotation.
- Performing further tests on similar low-cost, easily available, network-compatible microcontrollers.
- Reviewing why the black-white-black-white pattern occurs in some microcontrollers and the impact of the physical structure of the SRAM on this effect.
- Conduct identical tests on the RISC-V-based ESP32-C3 and compare the results with the existing data.

List of Figures

List of Tables

List of Listings

Acronyms

6LoWPAN	IPv6 over Low power Wireless Personal Area Network
AES	Advanced Encryption Standard
AES-GCM	AES Galois Counter Mode
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BCH	Bose-Chaudhuri-Hocquenghem Code
CBA	Certificate Based Authentication
CD	Compact Disk
CN	Carbon Nanotube
CNN	Cellular Neural Network
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CRP	Challenge Response Pair
DDS	Data Distribution Service
DegOI	Degree of Instability
DNS	Domain Name System
DRAM	Dynamic Random Access Memory
DTLS	Datagram Transport Layer Security
DuT	Device under Test

ECC	Error Correcting Code
ECC	Elliptic Curve Cryptography
EEPROM	Electrically Erasable Programmable Read-Only Memory
FPGA	Field Programmable Gate Array
GCM	Galois Counter Mode
GPIO	General Purpose Input/Output
HCI	Hot Carrier Injection
HDD	Hard Drive Disk
HEX	Hexadecimal
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transport Protocol Secure
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
ID	Identifier
IDE	Integrated Development Environment
IDM	Identity Management
idP	Identity Provider
IoT	Internet of Things
IP	Intellectual Property
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
JSON	JavaScript Object Notation

KB	Kilobyte
KDF	Key Derivation Function
LC	Inductor [symbolized by L] - Capacitor
LDPC	Low-Density Parity-Check
LoRaWAN	Long Range Wide Area Network
LTE	Long-Term Evolution
LVQ	Last Value Queue
MB	Megabyte
MECCA	MEmory Cell-based Chip Authentication
MEMS	Micro-Electrico-Mechanical System
MITM	Man-in-the-Middle
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MQTT	Message Queuing Telemetry Transport
mTLS	Mutual TLS
NASA	National Aeronautics and Space Administration
NBTI	Negative Bias Temperature Instability
NEMS	Nano-Electrico-Mechanical System
NFC	Near Field Communication
NVM	Non Volatile Memory
OTP	One Time Password
PBKDF2	Password Based Key Derivation Function 2
PCKGen	Phase change key generator
PKG	Private Key Generator
PKI	Public Key Infrastructure
PoC	Proof of Concept

PUF	Physical Unclonable Function
Q-EPUF	Quantum Electronic PUF
Q-OPUF	Quantum Optical PUF
RAM	Random-Access Memory
REST	representational state transfer
RF-DNA	Radio-Frequency DNA
RFID	Radio Frequency Identification
RNG	Random Number Generator
ROM	Read-Only Memory
RS	Reed-Solomon
RSA	Rivest-Shamir-Adleman
RTC	Real Time Clock
SHA-2	Secure Hash Algorithm 2
SHIC	Super High Information Content
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SRAM	Static Random-Access Memory
SRAM PUF	Static Random-Access Memory Physical Unclonable Function
SSD	Solid State Drive
SSID	Service Set IDentifier
SSL	Secure Socket Layer
STT-MRAM	Spin-Transfer-Torque Magnetic RAM
TCP	Transmission Control Protocol
TDDB	Electromigration and Time-Dependent Dielectric Breakdown

TERO	Transient Effect Ring Oscillator
TLS	Transport Layer Security
TPM	Trusted Platform Module
TV	Threshold Voltage
UDP	User Datagram Protocol
UID	unique identifier
ULP	Ultra Low Power
USB	Universal Serial Bus
VDD	Drain to Drain Voltage
VIA	Vertical Interconnect Access
VSS	Virtual Switching System (ground)
WAN	Wide Area Network
XMPP	Extensible Messaging and Presence Protocol

Bibliography

[1] Intrinsic ID, *Markets - Intrinsic ID | Home of PUF Technology -intrinsic-id.com*, https://www.intrinsic-id.com/markets/, [Accessed 06-02-2024].

[2] Paria Samadi Parviznejad, "The future of devices in digital businesses and improving productivity," in *Building Smart and Sustainable Businesses With Transformative Technologies*, IGI Global, 2024, pp. 16–37.

[3] Rodrigo Santos, Gabriel Eggly, Julián Gutierrez, and Carlos I Chesñevar, "Extending the iot-stream model with a taxonomy for sensors in sustainable smart cities," *Sustainability*, vol. 15, no. 8, p. 6594, 2023.

[4] Jetmir Haxhibeqiri, Eli De Poorter, Ingrid Moerman, and Jeroen Hoebeke, "A survey of lorawan for iot: From technology to application," *Sensors*, vol. 18, no. 11, p. 3995, 2018.

[5] Asif Ali Laghari, Kaishan Wu, Rashid Ali Laghari, Mureed Ali, and Abdullah Ayub Khan, "A review and state of art of internet of things (iot)," *Archives of Computational Methods in Engineering*, pp. 1–19, 2021.

[6] P.V. Dudhe, N.V. Kadam, R. M. Hushangabade, and M. S. Deshmukh, "Internet of things (iot): An overview and its applications," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017, pp. 2650–2653. DOI: 10.1109/ICECDS.2017.8389935.

[7] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi, "Internet of things (iot) communication protocols," in *2017 8th International conference on information technology (ICIT)*, IEEE, 2017, pp. 685–690.

[8] Sakina Elhadi, Abdelaziz Marzak, Nawal Sael, and Soukaina Merzouk, "Comparative study of iot protocols," *Smart Application and Data Analysis for Smart Cities (SADASC'18)*, 2018.

[9] Yahya Atwady and Mohammed Hammoudeh, "A survey on authentication techniques for the internet of things," in *Proceedings of the International Conference on Future Networks and Distributed Systems*, ser. ICFNDS '17, Cambridge, United Kingdom: Association for Computing Machinery, 2017, ISBN: 9781450348447. DOI: 10.1145/3102304.3102312. [Online]. Available: https://doi.org/10.1145/3102304.3102312.

[10] VL Shivraj, MA Rajan, Meena Singh, and P Balamuralidhar, "One time password authentication scheme based on elliptic curves for internet of things (iot)," in *2015 5th National Symposium on Information Technology: Towards New Smart World (NSITNSW)*, IEEE, 2015, pp. 1–6.

[11] René Hummen, Jan H Ziegeldorf, Hossein Shafagh, Shahid Raza, and Klaus Wehrle, "Towards viable certificate-based authentication for the internet of things," in *Proceedings of the 2nd ACM workshop on Hot topics on wireless network security and privacy*, 2013, pp. 37–42.

[12] Ola Salman, Sarah Abdallah, Imad H Elhajj, Ali Chehab, and Ayman Kayssi, "Identity-based authentication scheme for the internet of things," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, IEEE, 2016, pp. 1109–1111.

[13] Chi-Wei Lien and Sudip Vhaduri, "Challenges and opportunities of biometric user authentication in the age of iot: A survey," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–37, 2023.

[14] Nataliia Neshenko, Elias Bou-Harb, Jorge Crichigno, Georges Kaddoum, and Nasir Ghani, "Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations," *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2702–2733, 2019. DOI: 10.1109/COMST.2019.2910750.

[15] Espressif Systems, *Esp32 datasheet*, https://web.archive.org/web/20240118221803/https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf, Accessed: 2024-01-22, 2023.

[16] Bushra AlBelooshi, Ernesto Damiani, Khaled Salah, and Thomas Martin, "Securing cryptographic keys in the cloud: A survey," *IEEE Cloud Computing*, vol. 3, no. 4, pp. 42–56, 2016.

[17] Yansong Gao, Said F Al-Sarawi, and Derek Abbott, "Physical unclonable functions," *Nature Electronics*, vol. 3, no. 2, pp. 81–91, 2020.

[18] Thomas McGrath, Ibrahim E. Bagci, Zhiming M. Wang, Utz Roedig, and Robert J. Young, "A PUF taxonomy," *Applied Physics Reviews*, vol. 6, no. 1, p. 011 303, Feb. 2019, ISSN: 1931-9401. DOI: 10.1063/1.5079407. eprint: https://pubs.aip.org/aip/apr/article-pdf/

doi/10.1063/1.5079407/14575028/011303_1_online.pdf. [Online]. Available: https://doi.org/10.1063/1.5079407.

[19] Jae W Lee, Daihyun Lim, Blaise Gassend, G Edward Suh, Marten Van Dijk, and Srinivas Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No. 04CH37525)*, IEEE, 2004, pp. 176–179.

[20] Yida Yao, MyungBo Kim, Jianmin Li, Igor L Markov, and Farinaz Koushanfar, "Clockpuf: Physical unclonable functions based on clock networks," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2013, pp. 422–427.

[21] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 148–160.

[22] Lilian Bossuet, Xuan Thuy Ngo, Zouha Cherif, and Viktor Fischer, "A puf based on a transient effect ring oscillator and insensitive to locking phenomenon," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 30–36, 2013.

[23] Jason H Anderson, "A puf design for secure fpga-based embedded systems," in *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2010, pp. 1–6.

[24] Keith Lofstrom, W Robert Daasch, and Donald Taylor, "Ic identification circuit using device mismatch," in *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No. 00CH37056)*, IEEE, 2000, pp. 372–373.

[25] Duhyun Jeon, Jong Hak Baek, Dong Kyue Kim, and Byong-Deok Choi, "Towards zero bit-error-rate physical unclonable function: Mismatch-based vs. physical-based approaches in standard cmos technology," in *2015 Euromicro Conference on Digital System Design*, IEEE, 2015, pp. 407–414.

[26] Kyu-Man Hwang, Jun-Young Park, Hagyoul Bae, Seung-Wook Lee, Choong-Ki Kim, Myungsoo Seo, Hwon Im, Do-Hyun Kim, Seong-Yeon Kim, Geon-Beom Lee, *et al.*, "Nano-electromechanical switch based on a physical unclonable function for highly robust and stable performance in harsh environments," *ACS nano*, vol. 11, no. 12, pp. 12 547–12 552, 2017.

[27] Jonny Roberts, Ibrahim Ethem Bagci, MAM Zawawi, J Sexton, N Hulbert, YJ Noori, MP Young, CS Woodhead, Mohammed Missous, MA Migliorato, *et al.*, "Using quantum confinement to uniquely identify devices," *Scientific reports*, vol. 5, no. 1, p. 16 456, 2015.

[28] Fatemeh Tehranipoor, Nima Karimian, Wei Yan, and John A Chandy, "Dram-based intrinsic physically unclonable functions for system-level security and authentication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 1085–1097, 2016.

[29] André Schaller, Wenjie Xiong, Nikolaos Athanasios Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Stefan Katzenbeisser, and Jakub Szefer, "Intrinsic rowhammer pufs: Leveraging the rowhammer effect for improved security," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2017, pp. 1–7.

[30] Jorge Guajardo, Sandeep S Kumar, Geert-Jan Schrijen, and Pim Tuyls, "Fpga intrinsic pufs and their use for ip protection," in *Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9*, Springer, 2007, pp. 63–80.

[31] Ryan Helinski, Dhruva Acharyya, and Jim Plusquellic, "A physical unclonable function defined using power distribution system equivalent resistance variations," in *Proceedings of the 46th Annual Design Automation Conference*, 2009, pp. 676–681.

[32] Patrick Koeberl, Ünal Kocaba, and Ahmad-Reza Sadeghi, "Memristor pufs: A new generation of memory-based physically unclonable functions," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2013, pp. 428–431.

[33] Le Zhang, Zhi Hui Kong, and Chip-Hong Chang, "Pckgen: A phase change memory based cryptographic key generator," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2013, pp. 1444–1447.

[34] Le Zhang, Xuanyao Fong, Chip-Hong Chang, Zhi Hui Kong, and Kaushik Roy, "Highly reliable memory-based physical unclonable function using spin-transfer torque mram," in *2014 IEEE international symposium on circuits and systems (ISCAS)*, IEEE, 2014, pp. 2169–2172.

[35] Jorge Guajardo, Boris kori, Pim Tuyls, Sandeep S Kumar, Thijs Bel, Antoon HM Blom, and Geert-Jan Schrijen, "Anti-counterfeiting, key distribution, and key storage in an ambient world via physical unclonable functions," *Information Systems Frontiers*, vol. 11, pp. 19–41, 2009.

[36] Gerald DeJean and Darko Kirovski, "Rf-dna: Radio-frequency certificates of authenticity," in *Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9*, Springer, 2007, pp. 346–363.

[37] Jangbae Kim, Je Moon Yun, Jongwook Jung, Hyunjoon Song, Jin-Baek Kim, and Hyotcherl Ihee, "Anti-counterfeit nanoscale fingerprints based on randomly distributed nanowires," *Nanotechnology*, vol. 25, no. 15, p. 155 303, 2014.

[38] Zhen Chen, Yongbo Zeng, Gerald Hefferman, Yan Sun, and Tao Wei, "Fiberid: Molecular-level secret for identification of things," in *2014 IEEE international workshop on information forensics and security (WIFS)*, IEEE, 2014, pp. 84–88.

[39] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[40] Cheun Ngen Chong, Dan Jiang, Jiagang Zhang, and Long Guo, "Anti-counterfeiting with a random pattern," in *2008 second international conference on emerging security information, systems and technologies*, IEEE, 2008, pp. 146–153.

[41] Yameng Cao, Alexander J Robson, Abdullah Alharbi, Jonathan Roberts, Christopher S Woodhead, Yasir J Noori, Ramón Bernardo-Gavito, Davood Shahrjerdi, Utz Roedig, Vladimir I Falko, *et al.*, "Optical identification using imperfections in 2d materials," *2D Materials*, vol. 4, no. 4, p. 045 021, 2017.

[42] Ronald S Indeck and Marcel W Muller, *Method and apparatus for fingerprinting magnetic media*, US Patent 5,365,586, Nov. 1994.

[43] JD Key, "Some error-correcting codes and their applications," *Applied Mathematical Modeling: A Multidisciplinary Approach*, 1999.

[44] Claude Elwood Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[45] Robert J. McEliece, "The reliability of computer memories," *Scientific American*, vol. 252, no. 1, pp. 88–95, 1985, ISSN: 00368733, 19467087. [Online]. Available: `http://www.jstor.org/stable/24967550` (visited on 01/27/2024).

[46] *Digital revolution (part iii) - error correction codes*, `https://www.ams.org/publicoutreach/feature-column/fcarc-errors6`, Accessed: 2024-01-27, American Mathematical Society, 2023.

[47] Insah Bhurtah, Pierre Clarel Catherine, and K.M. Sunjiv Soyjaudah, "Enhancing the error-correcting performance of ldpc codes for lte and wifi," in *International Conference on Computing, Communication Automation*, 2015, pp. 1406–1410. DOI: `10.1109/CCAA.2015.7148600`.

[48] TOSHIBA CORPORATION, *Solid state drive for client applications*, Sep. 2014. [Online]. Available: https://europe.kioxia.com/content/dam/kioxia/en-europe/business/ssd/document/asset/KIE_TC_WP_201409-1_EN.pdf.

[49] Matthias Hiller, Ludwig Kürzinger, and Georg Sigl, "Review of error correction for pufs and evaluation on state-of-the-art fpgas," *Journal of Cryptographic Engineering*, vol. 10, Sep. 2020. DOI: 10.1007/s13389-020-00223-w.

[50] Electronics Notes, *What is sram memory: Static ram*. [Online]. Available: https://www.electronics-notes.com/articles/electronic_components/semiconductor-ic-memory/static-ram-sram.php.

[51] Wendong Wang, Adit D Singh, and Ujjwal Guin, "A systematic bit selection method for robust sram pufs," *Journal of Electronic Testing*, vol. 38, no. 3, pp. 235–246, 2022.

[52] Intrinsic ID, *Automotive Security - Intrinsic ID | Home of PUF Technology -intrinsic-id.com*, https://www.intrinsic-id.com/markets/automotive-security/, [Accessed 06-02-2024].

[53] Intrinsic ID, *Medical - Intrinsic ID | Home of PUF Technology - intrinsic-id.com*, https://www.intrinsic-id.com/markets/medical/, [Accessed 06-02-2024].

[54] Intrinsic ID, *Industrial IoT - Intrinsic ID | Home of PUF Technology - intrinsic-id.com*, https://www.intrinsic-id.com/markets/industrial-iot/, [Accessed 06-02-2024].

[55] Intrinsic ID, *Intrinsic-id.com*, http://www.intrinsic-id.com/wp-content/uploads/2020/10/White-Paper-Flexible-Key-Provisioning-with-SRAM-PUF.pdf, [Accessed 06-02-2024], Oct. 2020.

[56] Julian Dreyer, Ralf Tönjes, and Nils Aschenbruck, "Espuf enabling sram pufs on commodity hardware," in *2023 16th International Conference on Signal Processing and Communication System (ICSPCS)*, 2023, pp. 1–10. DOI: 10.1109/ICSPCS58109.2023.10261156.

[57] 2023. [Online]. Available: https://www.intrinsic-id.com/wp-content/uploads/2023/10/Intrinsic-ID-Fact-Sheet-2023-10-04.pdf.

[58] Intrinsic ID, *White paper - the reliability of sram puf - intrinsic id*, Aug. 2017. [Online]. Available: https://www.intrinsic-id.com/wp-content/uploads/2017/08/White-Paper-The-reliability-of-SRAM-PUF.pdf.

[59] B. M. S. Bahar Talukder, Farah Ferdaus, and Md Tauhidur Rahman, "Memory-based pufs are vulnerable as well: A non-invasive attack against sram pufs," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4035–4049, 2021. DOI: 10.1109/TIFS.2021.3101045.

[60] Zhi-Wei Lai and Kuen-Jong Lee, "Using unstable sram bits for physical unclonable function applications on off-the-shelf sram," in *2019 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2019, pp. 41–44. DOI: 10.1109/APCCAS47518.2019.8953143.

[61] PlatformIO, *Your gateway to embedded software development excellence*. [Online]. Available: https://platformio.org/.

[62] *Visual Studio Code - Code Editing. Redefined — code.visualstudio.com*, https://code.visualstudio.com, [Accessed 04-03-2024].

[63] Mar. 2023. [Online]. Available: https://randomnerdtutorials.com/esp32-esp8266-mysql-database-php.

[64] Cloudflare, [Accessed 07-03-2024], 2024. [Online]. Available: https://developers.cloudflare.com/cloudflare-one/identity/devices/access-integrations/mutual-tls-authentication/.

A. Appendix Visualisations

A.1. SRAM Visualisations

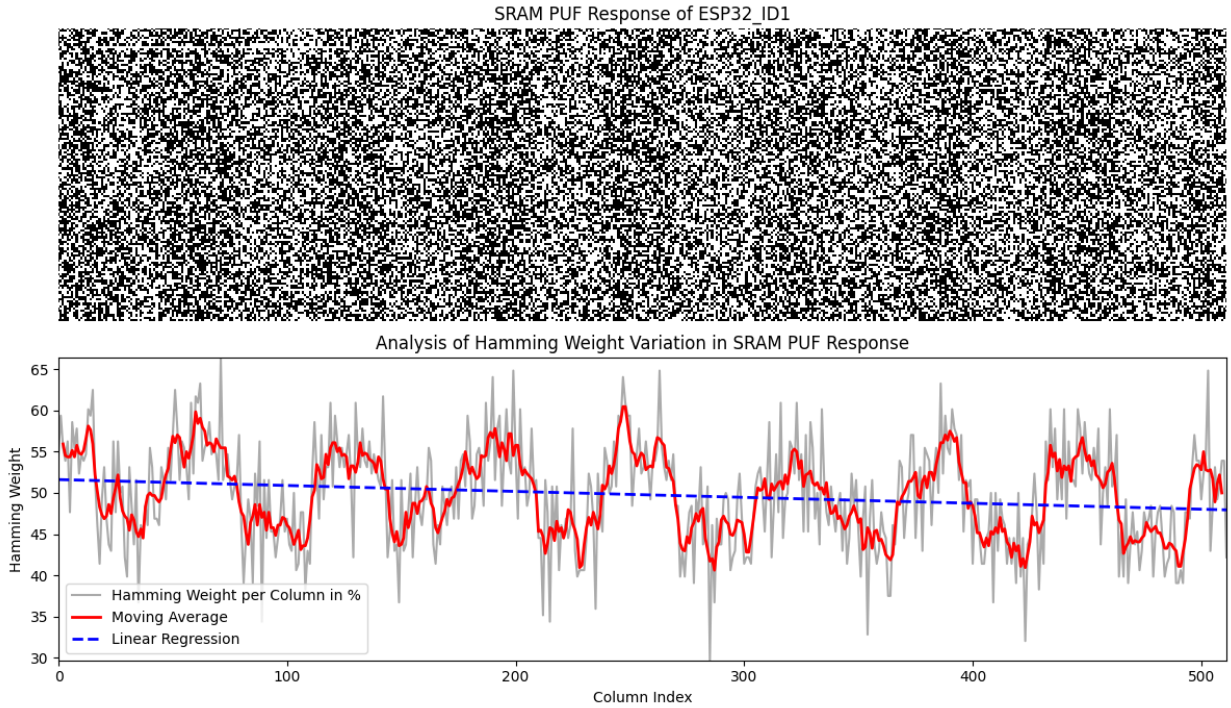

Figure A.1.: SRAM visualisation with moving average and linear regression (`ESP32_ID1`)

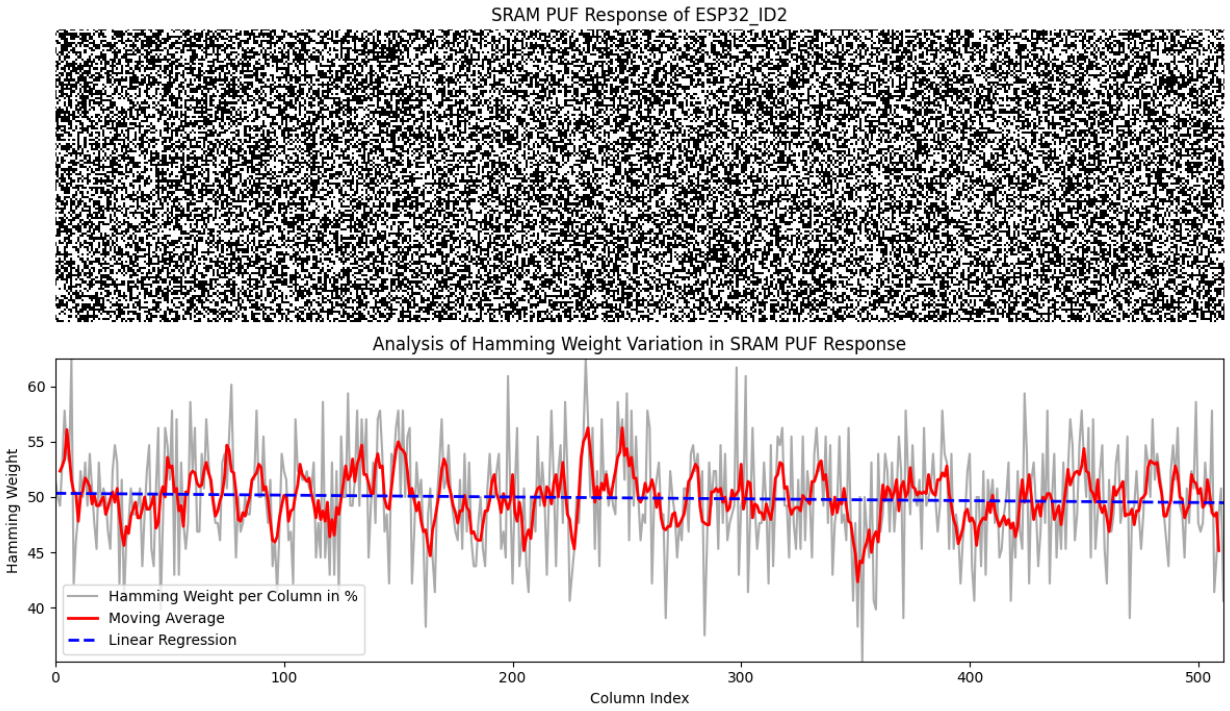

Figure A.2.: SRAM visualisation with moving average and linear regression (`ESP32_ID2`)

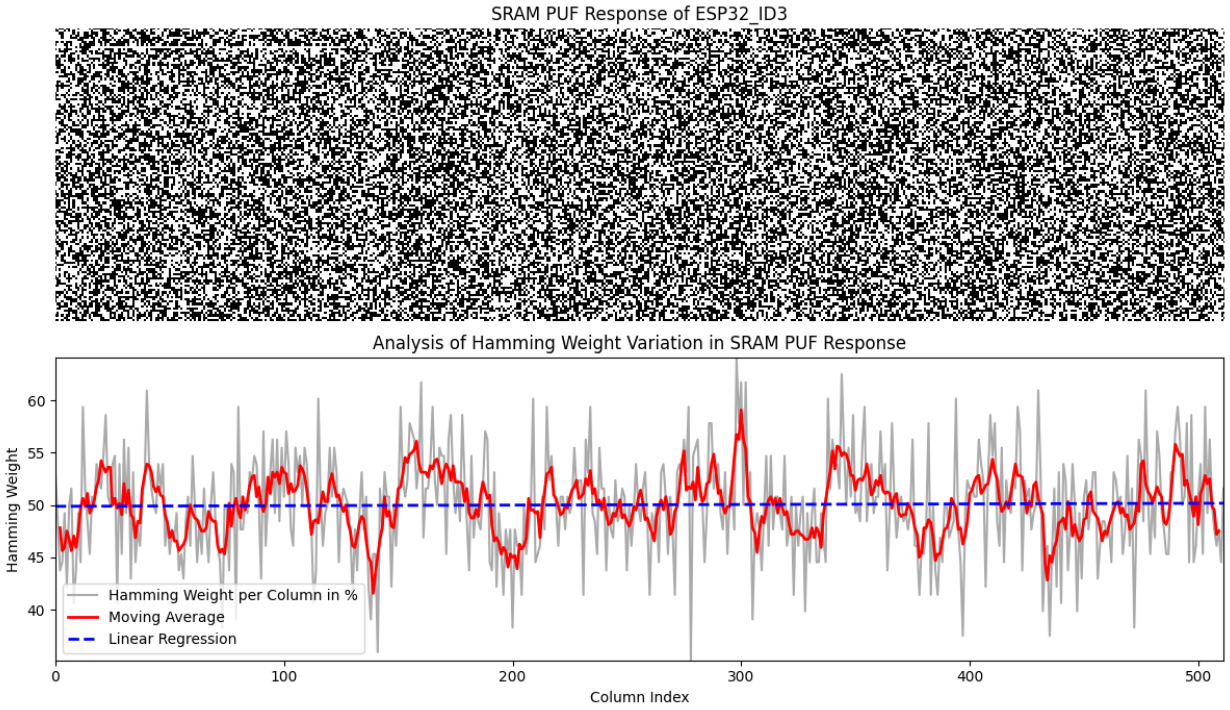

Figure A.3.: SRAM visualisation with moving average and linear regression (`ESP32_ID3`)

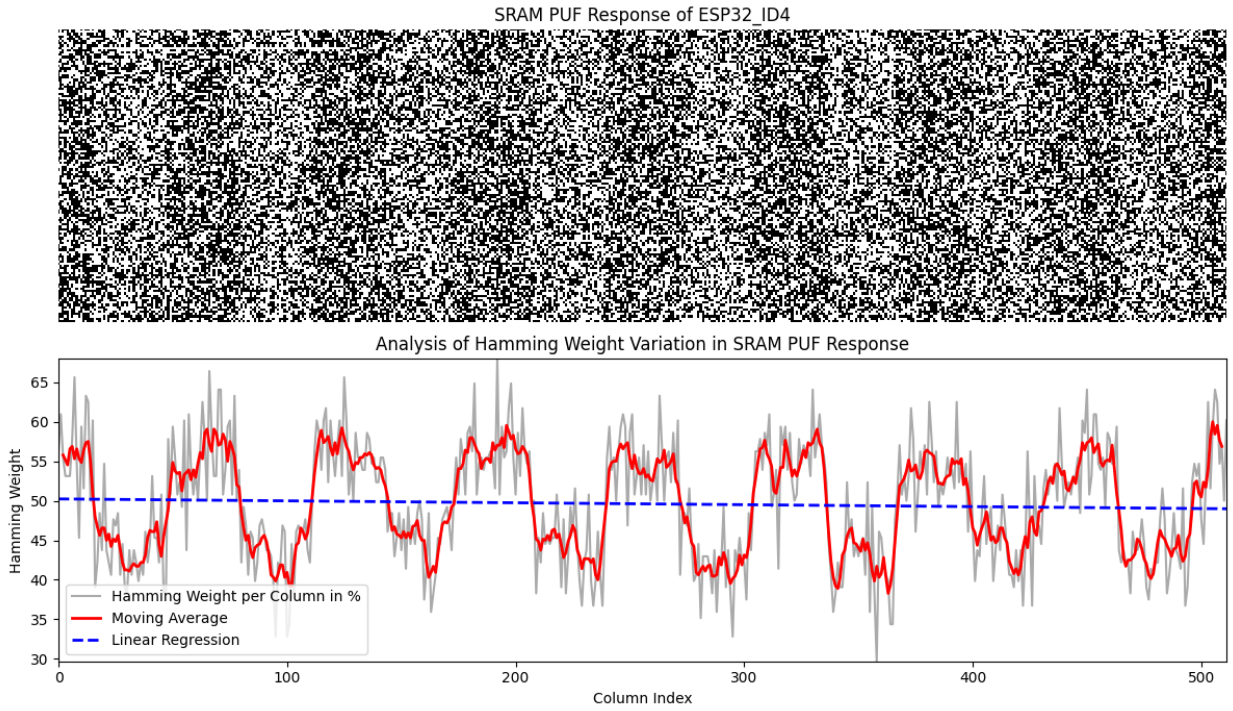

Figure A.4.: SRAM visualisation with moving average and linear regression (`ESP32_ID4`)

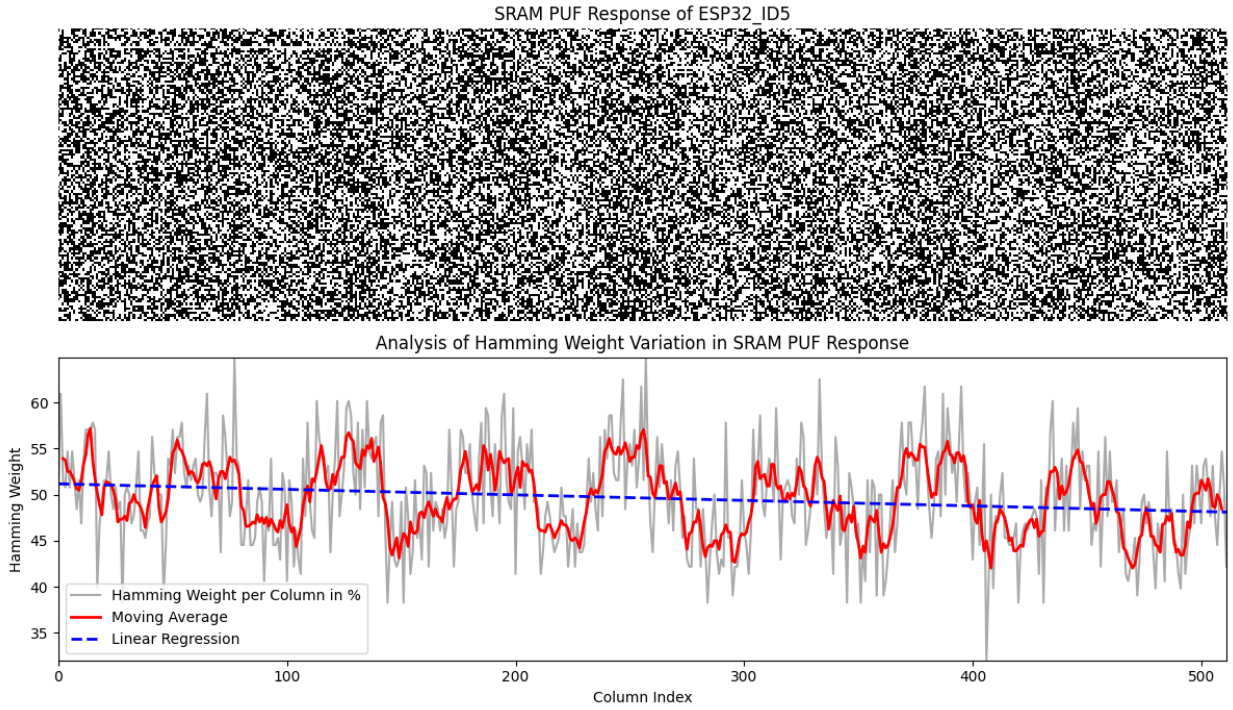

Figure A.5.: SRAM visualisation with moving average and linear regression (`ESP32_ID5`)

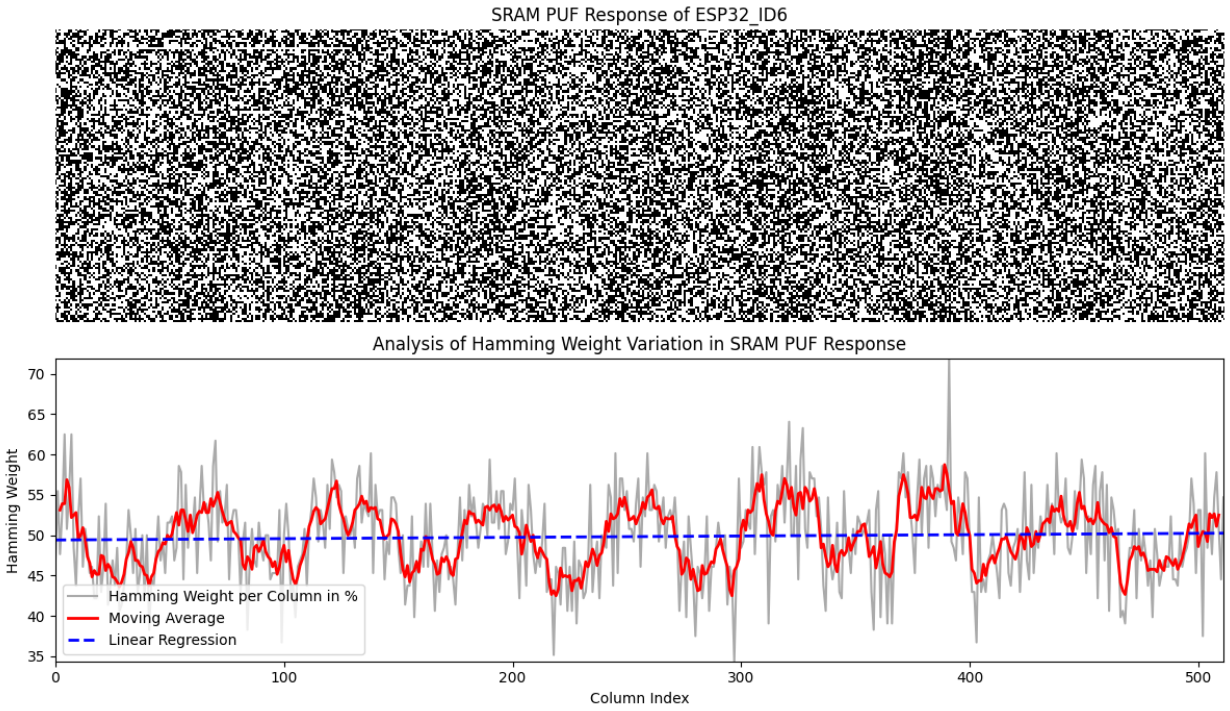

Figure A.6.: SRAM visualisation with moving average and linear regression (ESP32_ID6)

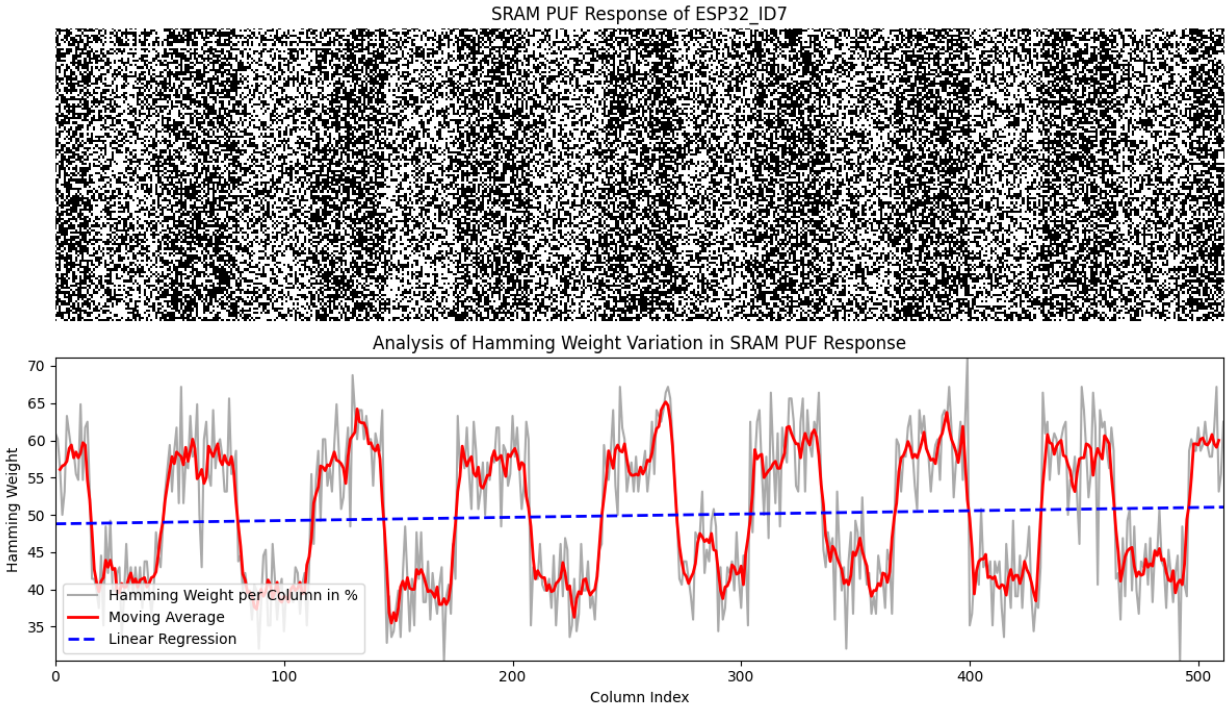

Figure A.7.: SRAM visualisation with moving average and linear regression (ESP32_ID7)

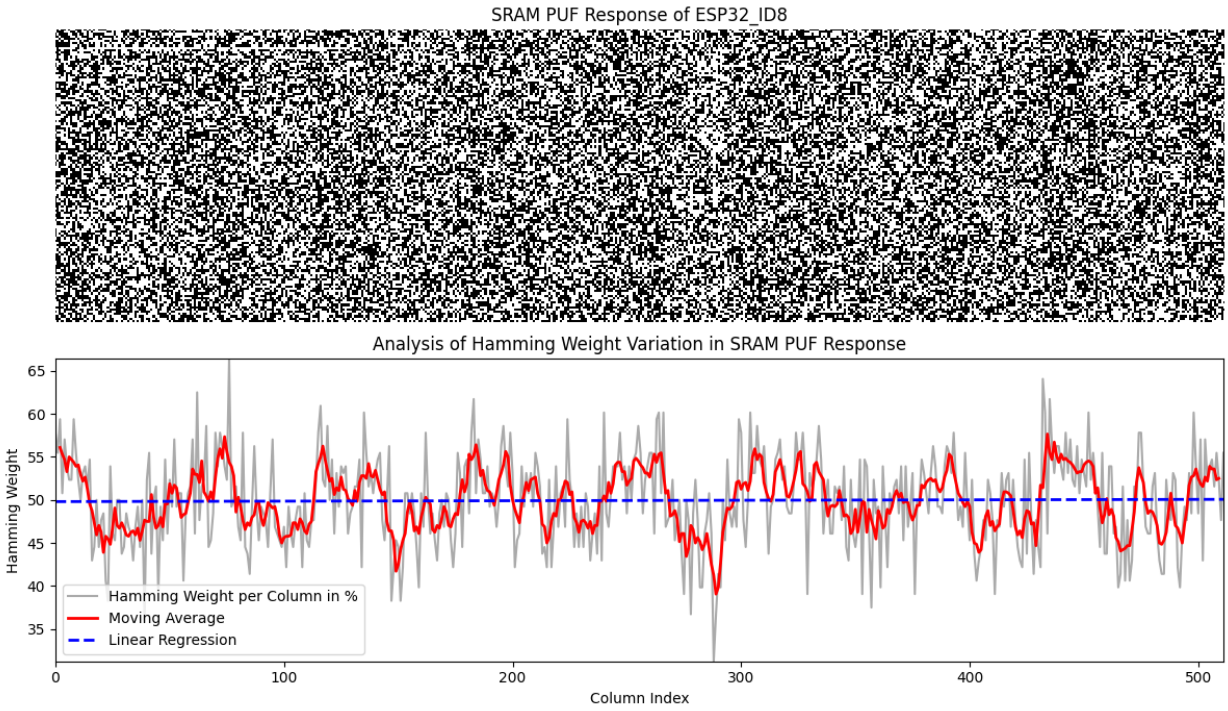

Figure A.8.: SRAM visualisation with moving average and linear regression (`ESP32_ID8`)

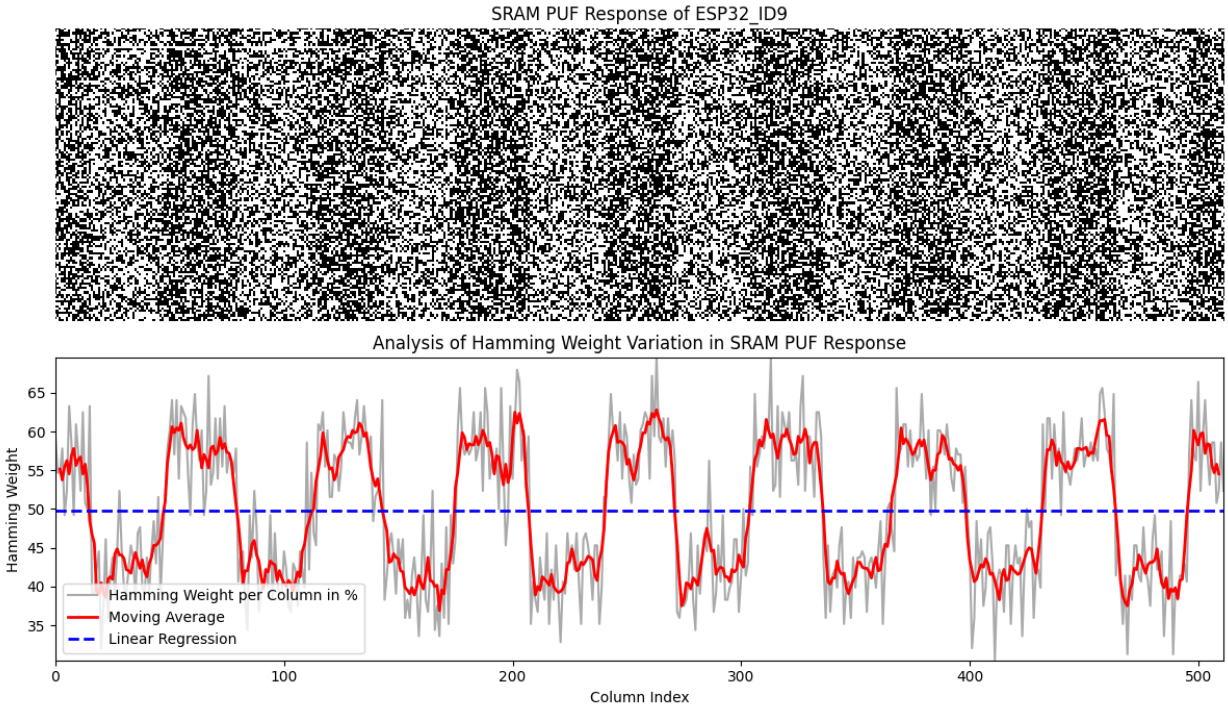

Figure A.9.: SRAM visualisation with moving average and linear regression (`ESP32_ID9`)

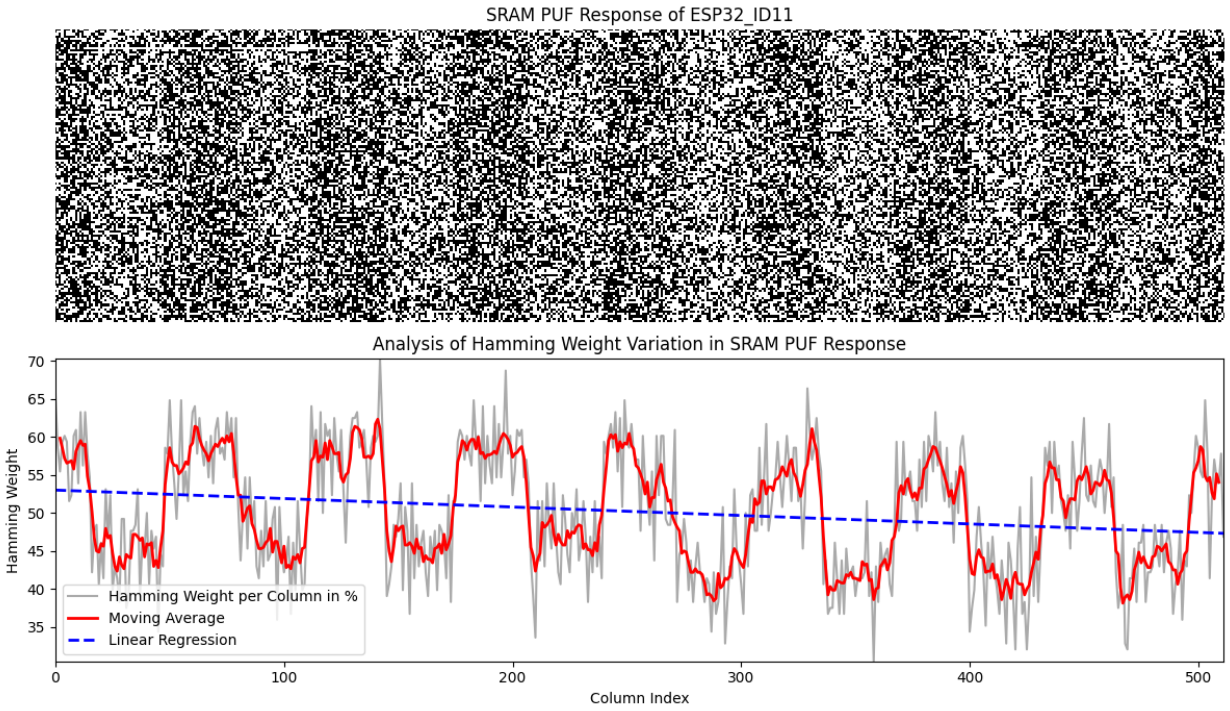

Figure A.10.: SRAM visualisation with moving average and linear regression (ESP32_ID11)

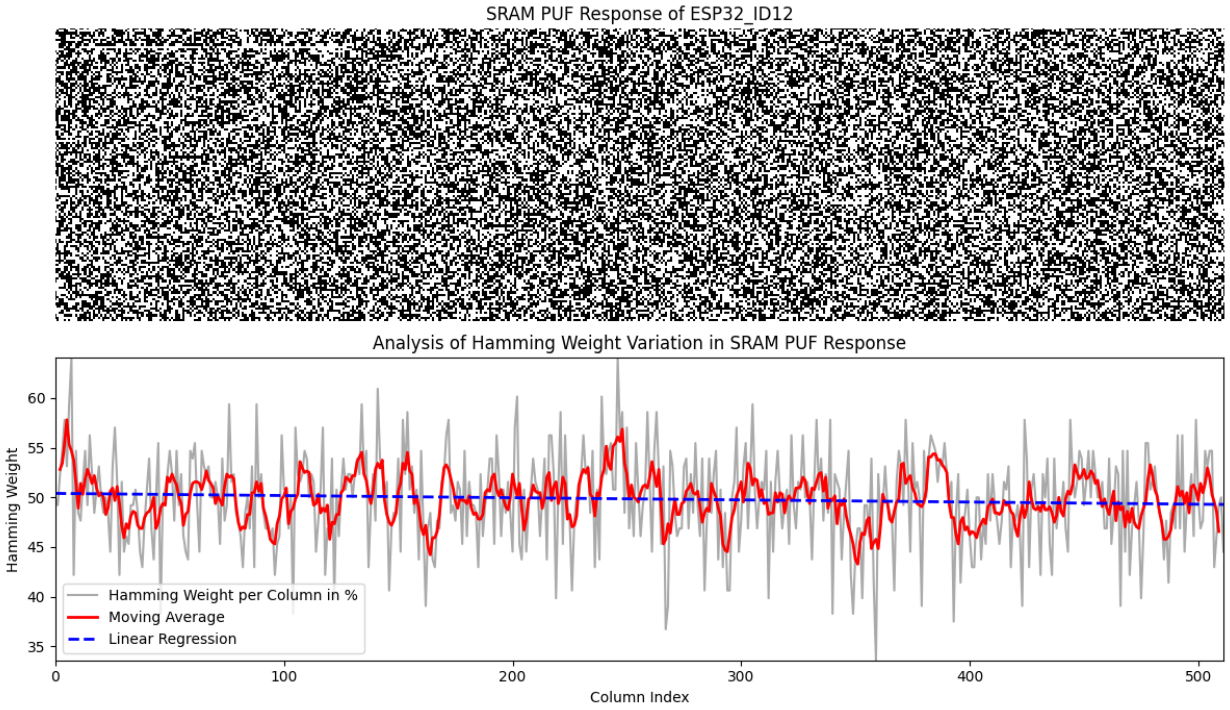

Figure A.11.: SRAM visualisation with moving average and linear regression (ESP32_ID12)

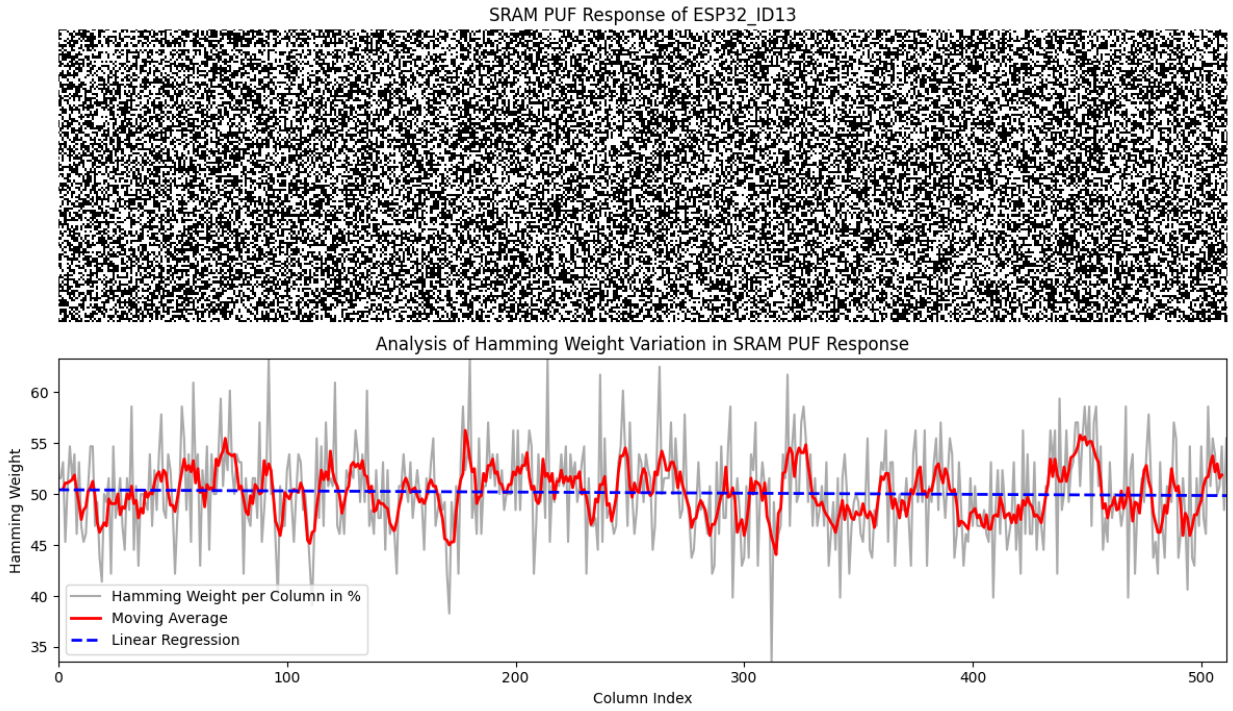

Figure A.12.: SRAM visualisation with moving average and linear regression (`ESP32_ID13`)

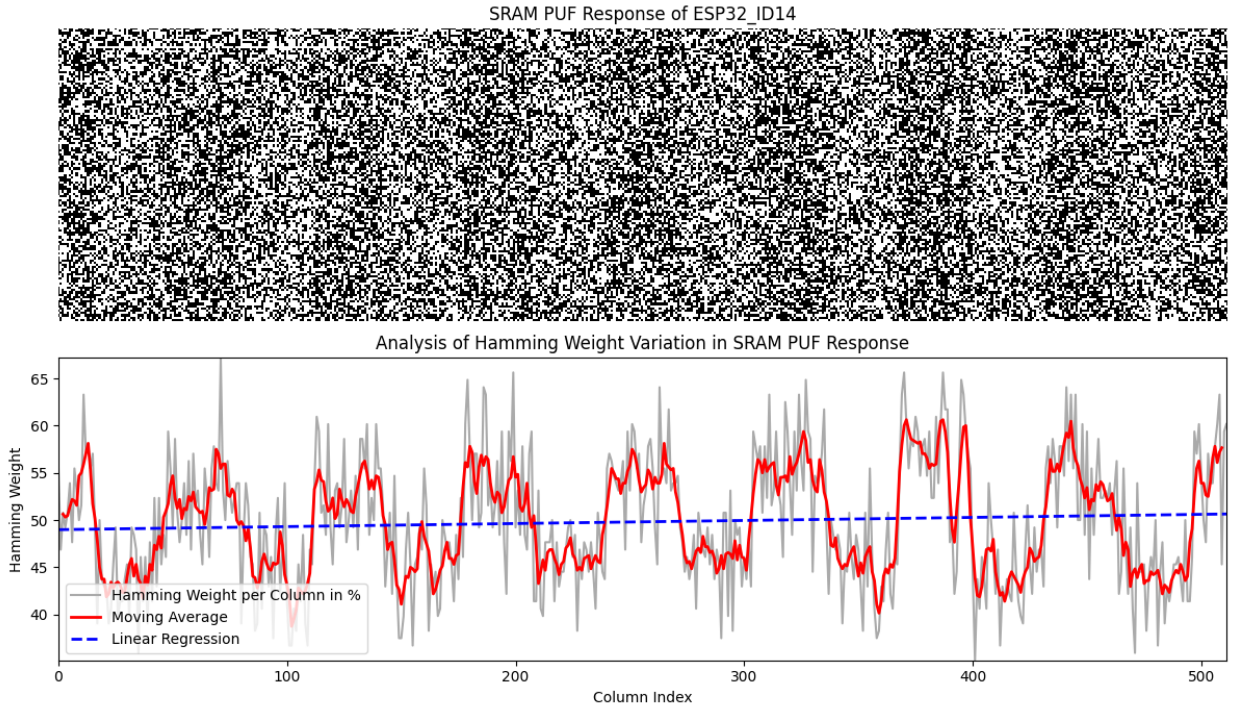

Figure A.13.: SRAM visualisation with moving average and linear regression (`ESP32_ID14`)

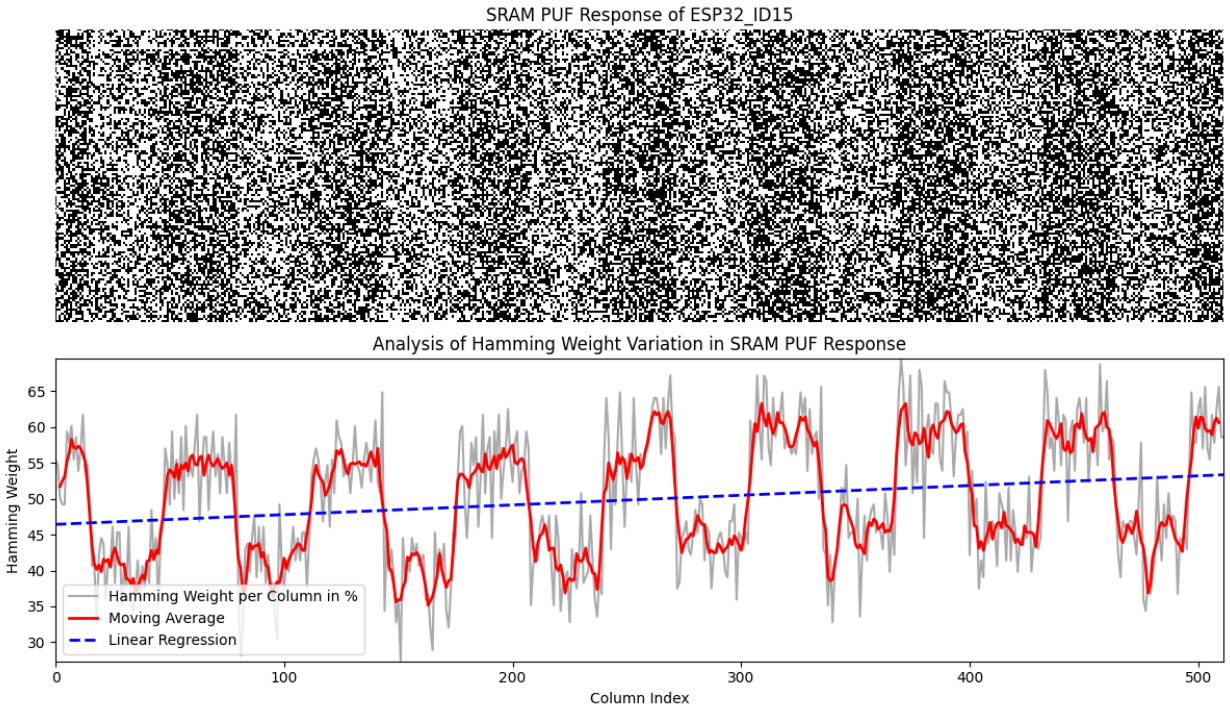

Figure A.14.: SRAM visualisation with moving average and linear regression (ESP32_ID15)

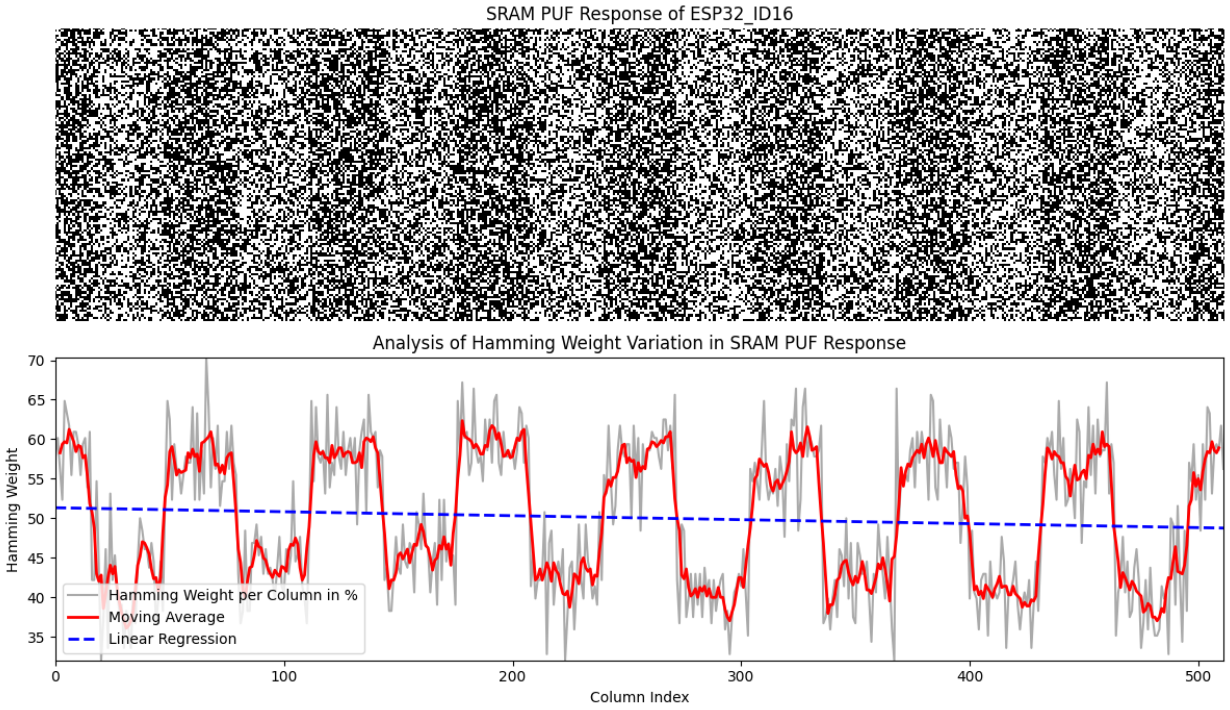

Figure A.15.: SRAM visualisation with moving average and linear regression (ESP32_ID16)

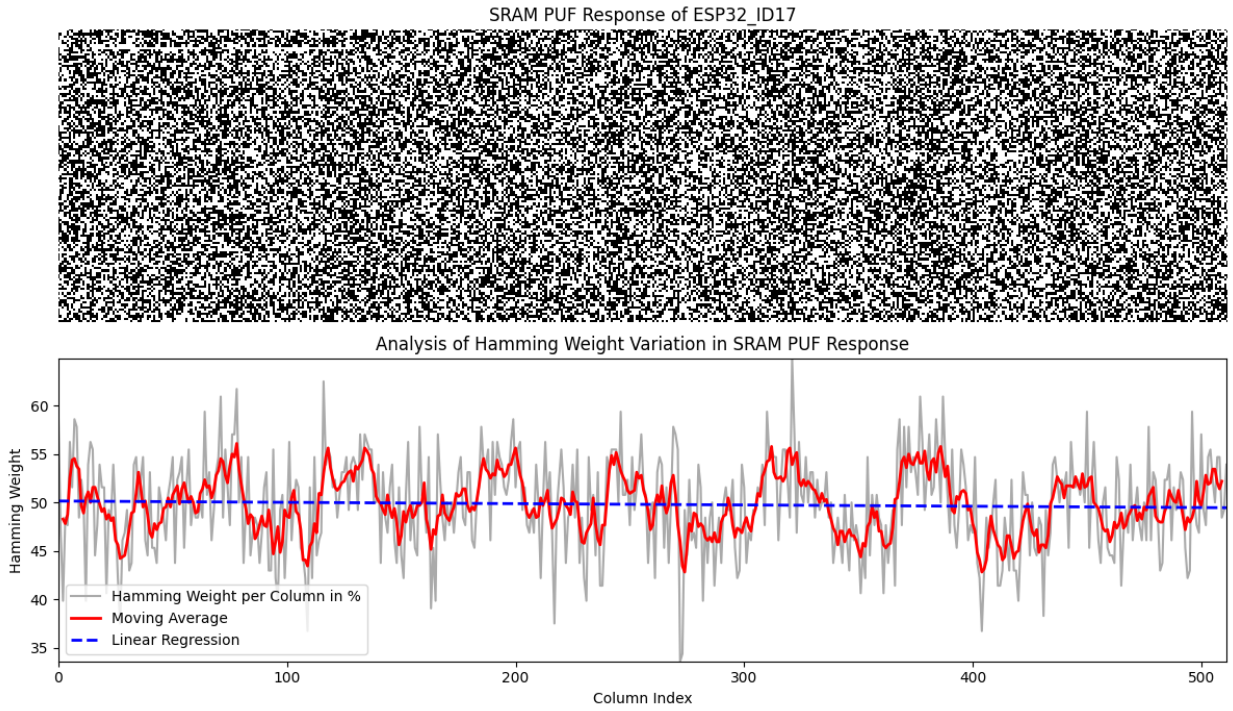

Figure A.16.: SRAM visualisation with moving average and linear regression (`ESP32_ID17`)

A.2. Hamming Distances Across Dataset

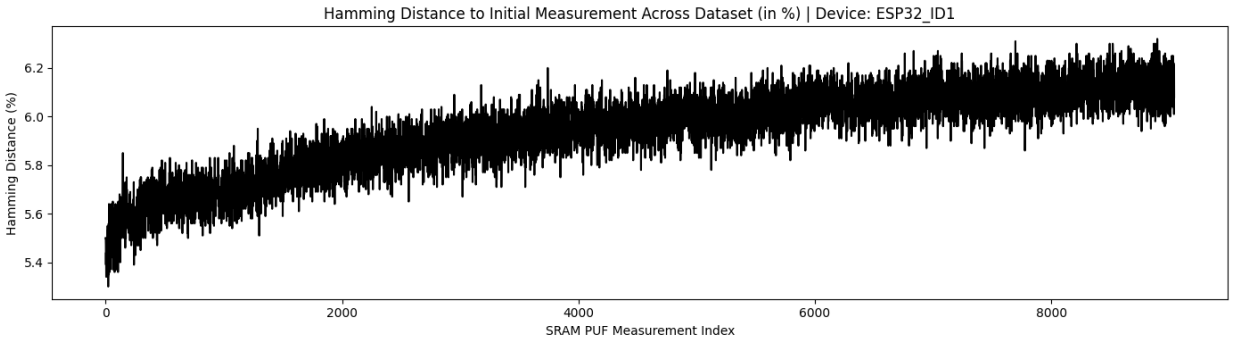

Figure A.17.: Hamming Distances from first measurement across dataset (`ESP32_ID1`)

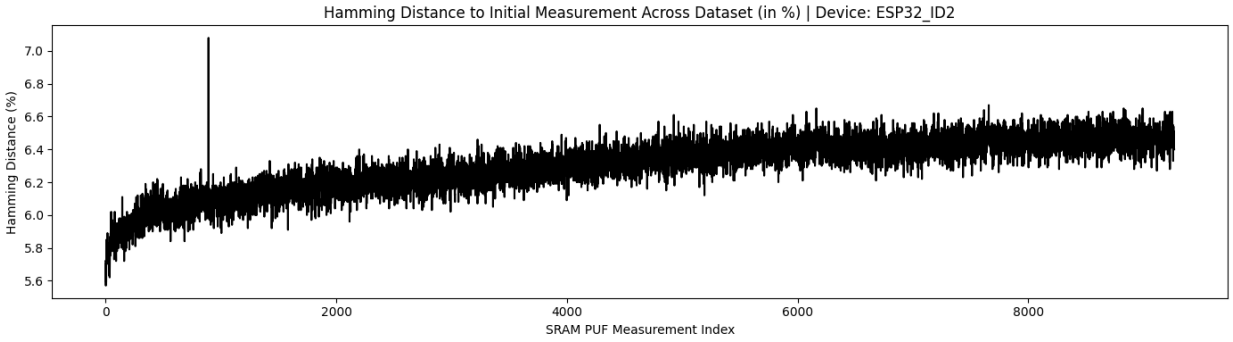

Figure A.18.: Hamming Distances from first measurement across dataset (`ESP32_ID2`)

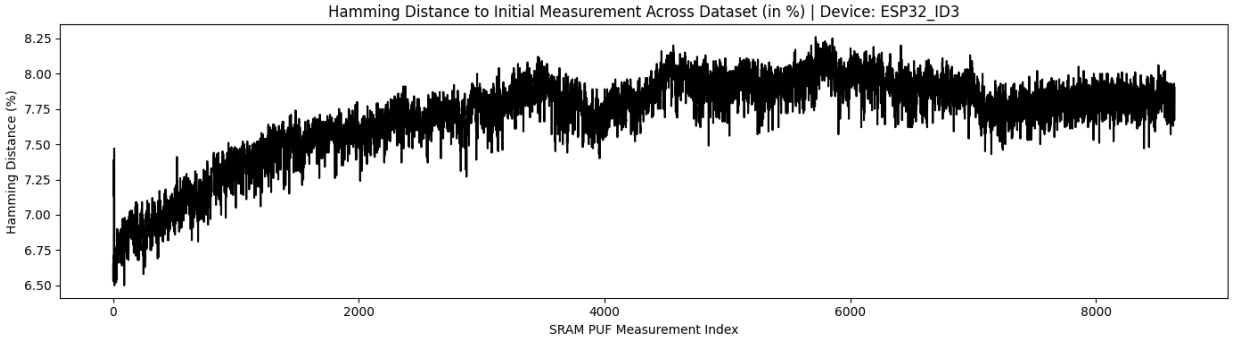

Figure A.19.: Hamming Distances from first measurement across dataset (`ESP32_ID3`)

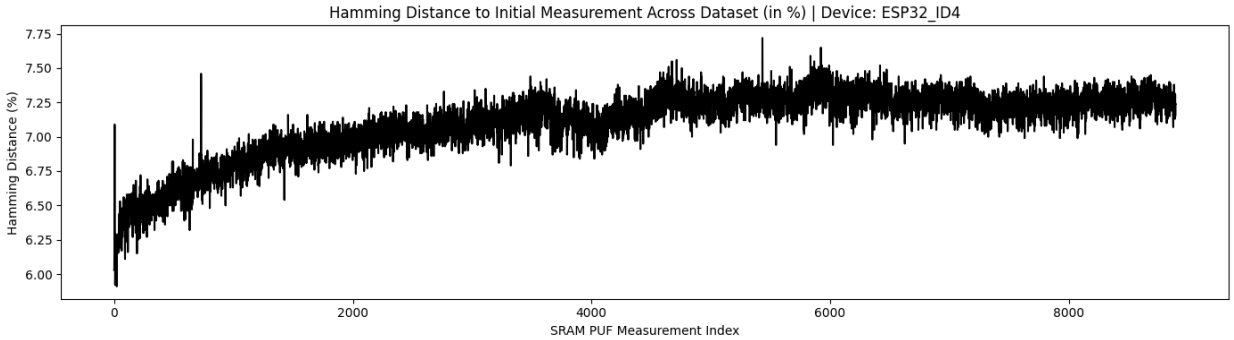

Figure A.20.: Hamming Distances from first measurement across dataset (`ESP32_ID4`)

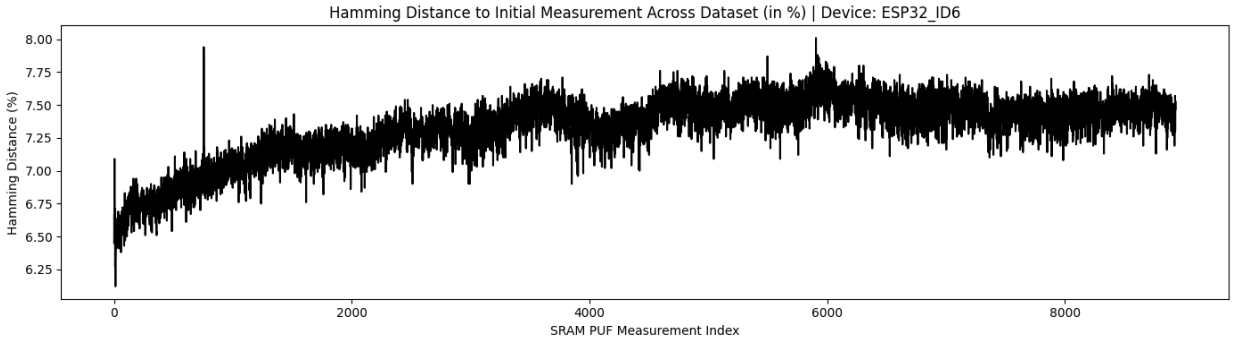

Figure A.21.: Hamming Distances from first measurement across dataset (`ESP32_ID6`)

A.3. Consecutive Hamming Distances

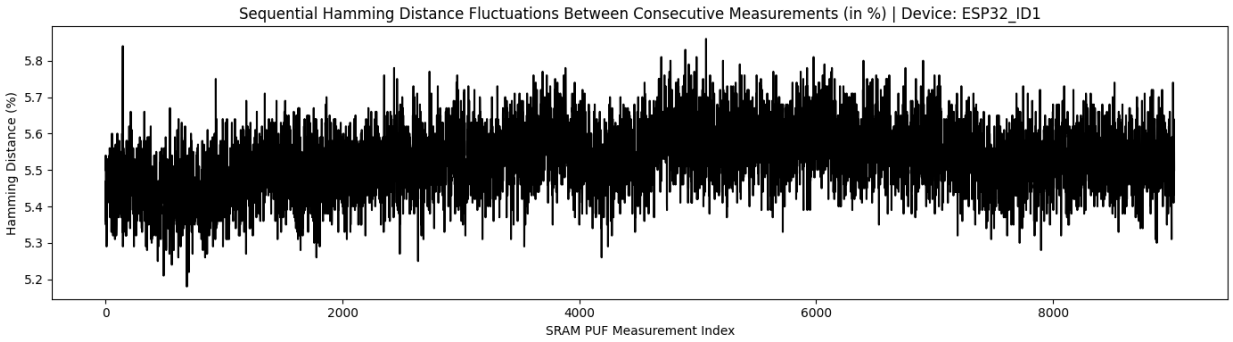

Figure A.22.: Consecutive Hamming Distances across dataset (`ESP32_ID1`)

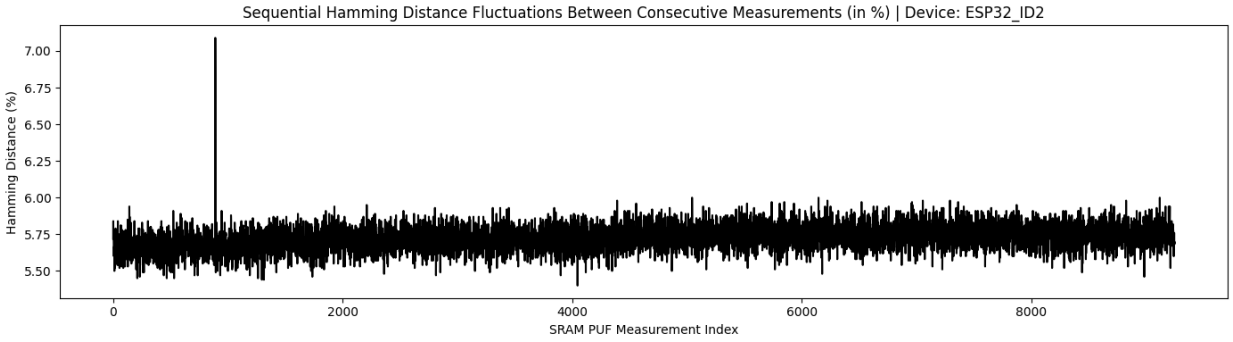

Figure A.23.: Consecutive Hamming Distances across dataset (`ESP32_ID2`)

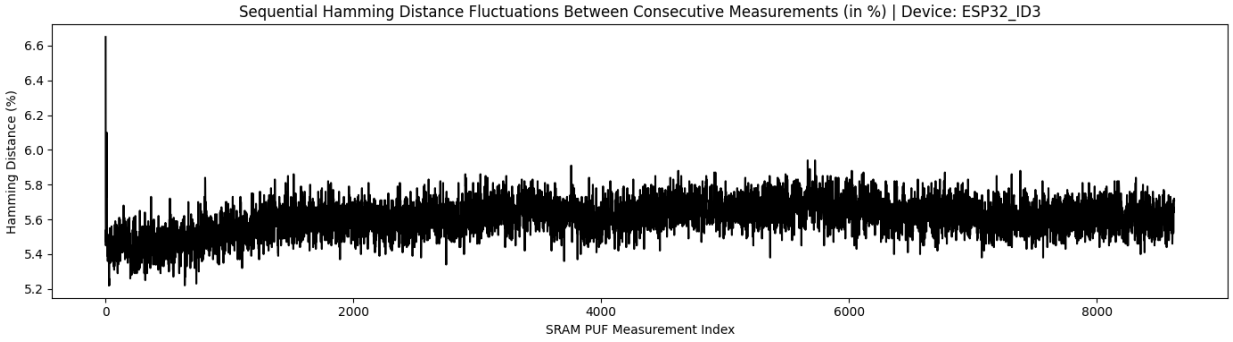

Figure A.24.: Consecutive Hamming Distances across dataset (`ESP32_ID3`)

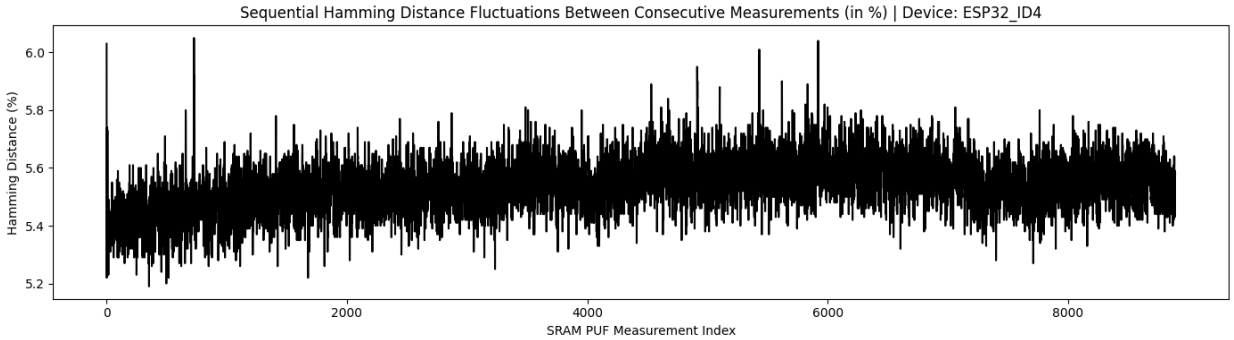

Figure A.25.: Consecutive Hamming Distances across dataset (`ESP32_ID4`)

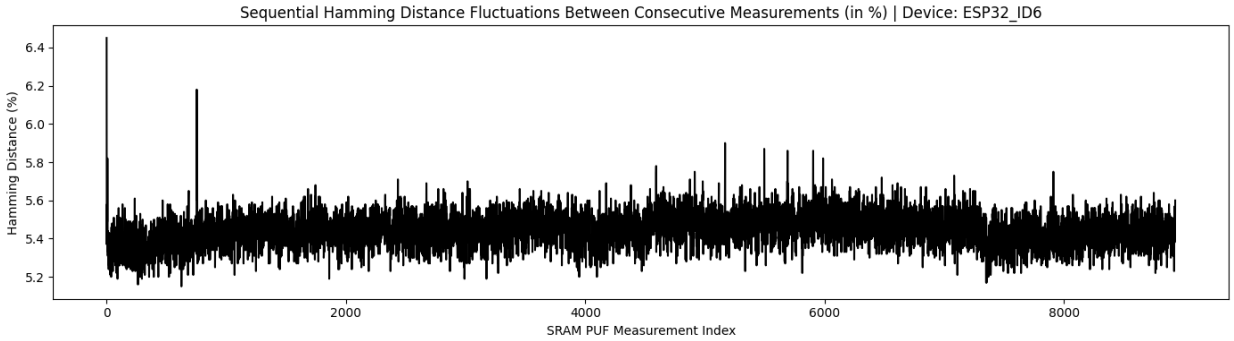

Figure A.26.: Consecutive Hamming Distances across dataset (`ESP32_ID6`)

A.4. Hamming Weight Across Dataset

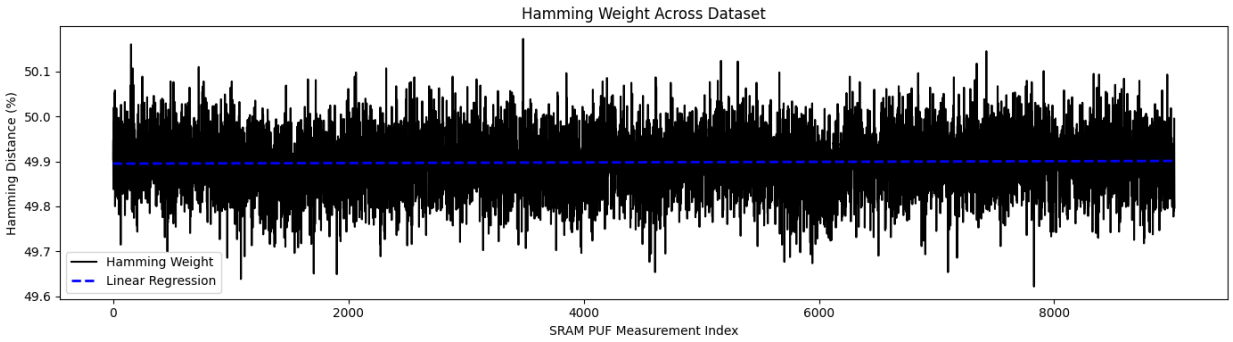

Figure A.27.: Hamming Weight visualisation across dataset (`ESP32_ID1`)

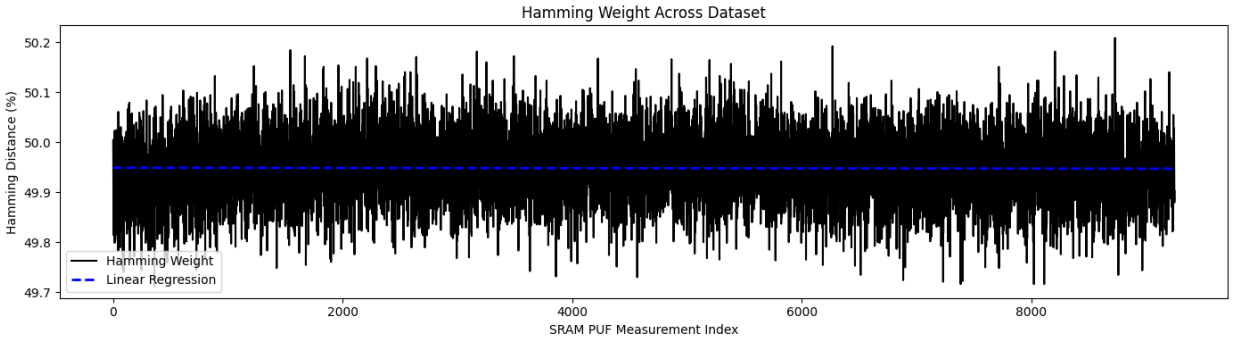

Figure A.28.: Hamming Weight visualisation across dataset (`ESP32_ID2`)

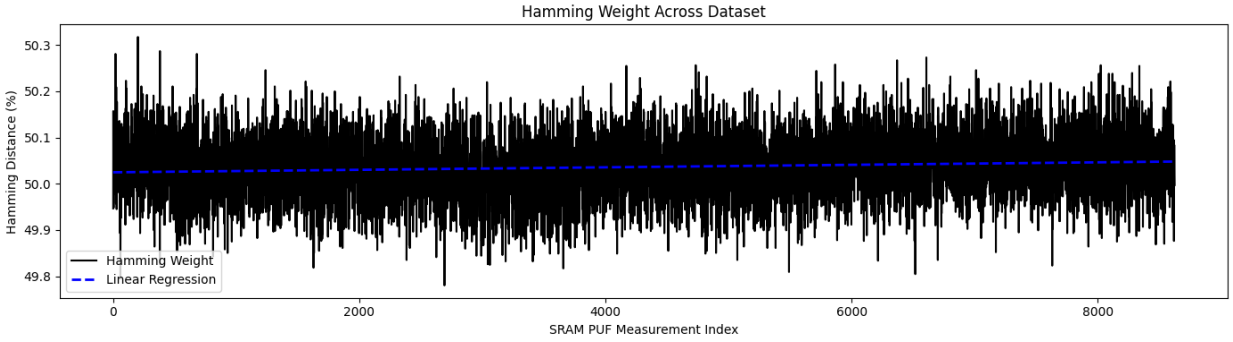

Figure A.29.: Hamming Weight visualisation across dataset (`ESP32_ID3`)

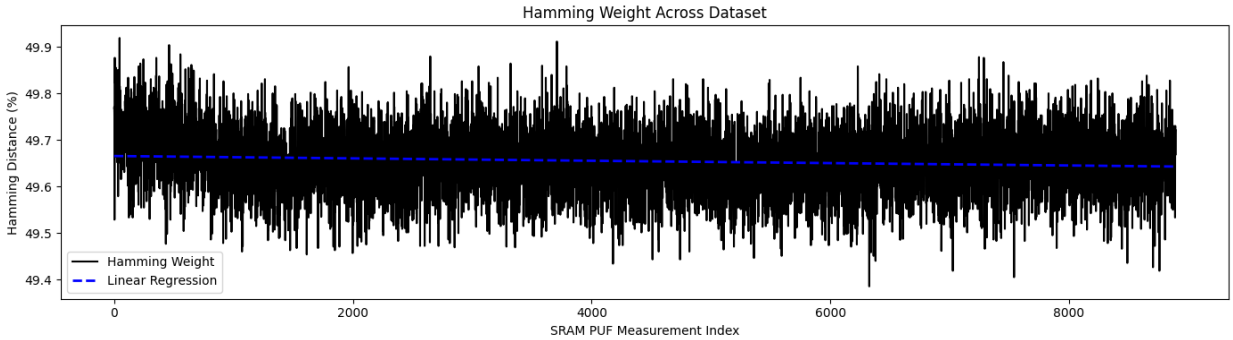

Figure A.30.: Hamming Weight visualisation across dataset (`ESP32_ID4`)

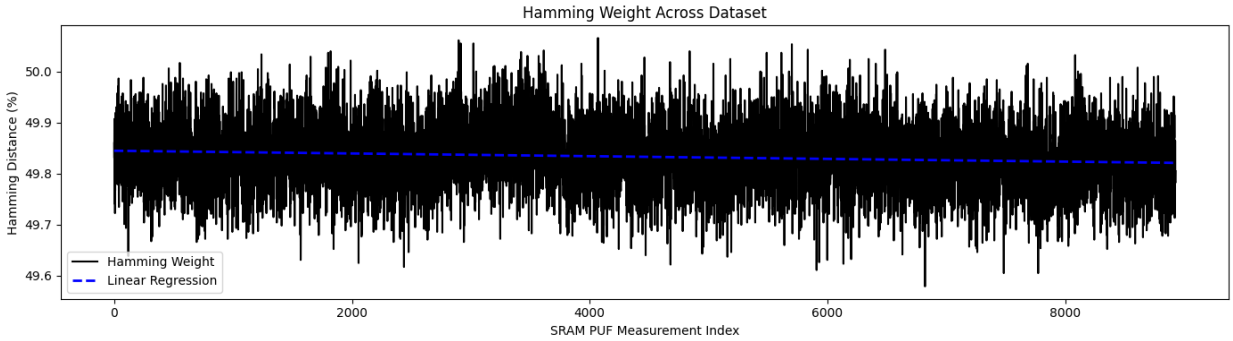

Figure A.31.: Hamming Weight visualisation across dataset (`ESP32_ID6`)

B. Appendix Source Code

Listing 17 provides an overview of all source code files that were referenced in this work and are provided on the CD.

```
./analysis
./analysis/hammingweightanddistance.py
./analysis/hammingdistance_table.py
./esp32
./esp32/read_sram_and_upload_to_measurementserver.cpp
./esp32/read_sram.cpp
./measurement_server
./measurement_server/database.js
./measurement_server/Dockerfile
./measurement_server/index.js
./measurement_server/package.json
./measurement_server/db
./measurement_server/docker-compose.yml
./authentication
./authentication/Dockerfile
./authentication/pufchallenge.py
./authentication/esp32_authenticate.cpp
./authentication/app.js
```

Listing 17: Overview of source code files on the CD