FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Physical unclonable functions on ESP32
Student:	Ondřej Staníček
Supervisor:	Ing. Jiří Buček, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2022/2023

Instructions

Physical unclonable functions are an emerging cryptographic primitive that can be used for device identification, authentication, and key generation in digital devices.

* Study the topic of physical unclonable functions (PUFs). Focus primarily on SRAM PUF.
* Design and implement a simple SRAM based PUF on a suitable ESP32 microcontroller.
* Experiment with the microcontroller's internal registers to find a suitable mechanism for SRAM power state control.
* Test the function of the PUF on several devices.
* Evaluate relevant PUF parameters (especially bit stability and uniqueness).
* Evaluate the distribution of SRAM bit values depending on operating temperature and power-off time.

Electronically approved by prof. Ing. Pavel Tvrdík, CSc. on 10 January 2022 in Prague.

Bachelor's thesis

PHYSICAL UNCLONABLE FUNCTIONS ON ESP32

Ondřej Staníček

Faculty of Information Technology
Department of Computer Systems
Supervisor: Ing. Jiří Buček, Ph.D.
May 2, 2022

Contents

List of Figures

List of Tables

List of code listings

Chtěl bych poděkovat především sit amet, consectetuer adipiscing elit. Curabitur sagittis hendrerit ante. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Sed vel lectus. Donec odio tempus molestie, porttitor ut, iaculis quis, sem. Suspendisse sagittis ultrices augue.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for non-profit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 2, 2022 .....

Abstract

This thesis analyzes the possibility of implementing a static random access memory (SRAM) physical unclonable function (PUF) on the ESP32 microcontroller.

First, literature research on the topic of PUFs is provided with focus on SRAM PUFs. A discussion on which properties the SRAM PUFs possess is presented.

Two power-control methods of SRAM memory on the ESP32 are proposed. An analysis of behavior of startup SRAM bit values depending on operating temperature and power-off time is conducted for both methods. Their suitability for the PUF implementation is discussed based on the experimental results.

Then, an implementation of SRAM PUF with stable response reconstruction is presented. Two different bit preselection methods are tested and a simple repetition error correction code (ECC) is used to stabilize the responses. The presented PUF design combines the two power-control methods to achieve faster and more reliable response extraction.

Reliability testing revealed that it is possible to reach 100 % success rate of response reconstruction across the temperature range of -40 to +70 °C. The responses can be used as cryptographic keys to secure the ESP32 platform. Finally, the proposed PUF design is implemented in an easy-to-use ESP32 library.

Keywords ESP32, physical unclonable functions, static random access memory, key generation, error correction codes, hardware security

Abstrakt

Tato práce zkoumá možnost implementace fyzicky neklonovatelné funkce (PUF) na základě statické paměti (SRAM) na mikrokontroléru ESP32.

Nejprve je provedena rešerše o problematice PUF se zaměřením na SRAM PUF. Také jsou popsány vlastnosti SRAM PUFu.

Jsou představeny dvě metody kontroly napájení SRAM paměti. Pro obě je provedena analýza chování neinicializovaných SRAM dat dle operační teploty a doby vypnutí. Na základě těchto výsledků je posouzena vhodnost jejich využití.

Poté je představena implementace SRAM PUFu se spolehlivou rekonstrukcí odpovědi. Dvě různé metody výběru stabilních bitů a opakovací samoopravný kód jsou použity ke stabilizaci odpovědí. Tato implementace kombinuje obě představené metody kontroly napájení paměti, čímž dosahuje rychlejší a spolehlivější extrakce odpovědí.

Testování spolehlivosti odhalilo, že je možné dosáhnout 100 % úspěšnosti rekonstrukce odpovědí při teplotách od -40 do +70 °C. Tyto odpovědi je možné použít jako kryptografické klíče k zabezpečení ESP32. Nakonec je tento návrh SRAM PUFu implementován jako snadno použitelná ESP32 knihovna.

Klíčová slova ESP32, fyzicky neklonovatelné funkce, statická paměť, samoopravné kódy, generování klíčů, hardwarová bezpečnost

List of Abbreviations

AES	Advanced Encryption Standard
API	application programming interface
BCH	Bose–Chaudhuri–Hocquenghem
BLE	Bluetooth Low Energy
CMOS	complementary metal–oxide–semiconductor
CPU	central processing unit
CTW	context-tree weighting
ECC	error correction code
FAR	false acceptance rate
FPGA	field-programmable gate array
FRR	false rejection rate
GPIO	general-purpose input/output
HD	Hamming distance
HW	Hamming weight
IoT	Internet of things
MAC	message authentication code
MCU	microcontroller unit
MOSFET	metal–oxide–semiconductor field-effect transistor
NBTI	negative-bias temperature instability
NIST	National Institute of Standards and Technology
NVS	non-volatile storage
PLL	phase-locked loop
POK	physically obfuscated key
POWF	physical one-way function
PRF	physical random function
PUF	physical unclonable function
ROM	read only memory
ROPUF	ring oscillator physical unclonable function
RTC	real-time clock
RTOS	real-time operating system
SoC	system on a chip
SPI	Serial Peripheral Interface
SRAM	static random access memory
TMV	temporal majority voting
TRNG	true random number generator
UART	universal asynchronous receiver-transmitter
ULP	ultra low power

Introduction

Motivation

Internet of things (IoT) is a fast-growing field of the information age that we live in right now. Millions of cheap and connected devices are being used in a huge number of applications such as smart homes, self-driving cars, smart cities and wearables.

As these devices are connected to the Internet and collect potentially sensitive data, it is crucial to make them as secure as possible. One of the key challenges of security is the generation and storage of cryptographic keys since they play a vital role in most cryptographic algorithms used to secure the devices.

Usually, the keys are randomly generated and stored in non-volatile memory. However, the non-volatile memory needs to be secured (even from invasive physical attacks) and the generation process must be unpredictable. Satisfying both of those requirements is extremely hard and expensive.

Physical unclonable functions (PUFs) try to solve this issue. They are based on physical properties which are inherently random and unique for each device. A PUF can be compared to a device's fingerprint. The cryptographic key is not stored digitally anywhere on the device but is reconstructed from the unique physical properties. PUFs can also be used for other tasks such as identification or authentication.

One example of an especially popular IoT platform is ESP32. It consists of a family of cheap microcontrollers and software to enable easy application development. The main goal of this thesis is to analyze the possibility of implementing a particular type of PUF, the static random access memory (SRAM) PUF, on this platform.

Objectives

The objectives of this thesis are the following:

- Analyze the topic of physical unclonable functions with a focus on SRAM PUF.
- Find a suitable mechanism for SRAM power state control on the ESP32 microcontroller.
- Design and implement a simple SRAM based PUF on the ESP32 microcontroller.
- Evaluate relevant PUF parameters such as bit stability and uniqueness on the resulting PUF implementation.
- Evaluate the behaviour of the PUF depending on operating temperature and SRAM power-off time.

- Test the function of the resulting PUF implementation on several devices.

Thesis structure

The thesis is divided into eight chapters in the following way:

1. Physical unclonable functions:
First, a description of a PUF is given in this chapter. Then, the main properties and classes of PUFs are outlined and parameters used to evaluate the performance of PUFs are defined. Lastly, concrete PUF applications and implementations are explained shortly.

2. SRAM PUF:
A thorough explanation of a SRAM PUF is given. The chapter then provides a discussion on which properties the SRAM PUF possesses and a brief introduction to silicon aging which affects this PUF implementation. At the end, existing research of SRAM PUFs on the ESP32 platform is summarized.

3. ESP32 platform:
An introduction to the ESP32 platform is provided in this chapter. A brief hardware and software description of the ESP32 microcontroller is given with a focus on parts that are important to the PUF implementation.

4. SRAM PUF implementation on ESP32:
Two ways of SRAM power state control on ESP32 are proposed in this chapter. The startup memory values obtained by the two methods are then analyzed based on operating temperature and memory power-off time. Some of the evaluation parameters defined in Chapter 1 are also used during the analysis.

5. Reliable PUF response reconstruction:
This chapter proposes a method to extract PUF responses reliably. This enables the PUF to be used for cryptographic key generation. In order to achieve this goal, stable bit preselection and error correction codes (ECCs) are used. At the end, the PUF is tested on 16 different devices during different operating temperatures.

6. ESP32 SRAM PUF library:
Based on the analyses in the previous chapters a simple SRAM PUF implementation is provided in an ESP32 library. Its functionality is explained in this chapter.

Chapter 1

Physical unclonable functions

1.1 PUF description

As PUFs are the main subject of this thesis, it is important to provide a thorough explanation of what a PUF is, what types and classes exist and what are the applications of this technology.

Since more and more types of PUFs are being invented, it turns out that creating a generalizable description is not a straightforward task. A dictionary definition of a PUF could be written as: "a PUF is an expression of an inherent and unclonable instance-specific feature of a physical object". One can imagine a PUF being an object's fingerprint in a comparable way to how humans have their own unique fingerprints. [1]

The first concept of PUF was proposed by Pappu in 2001. He used the term physical one-way function (POWF), which he described as a function operating on a physical system that could be easily computed but not easily inverted.[2] The first mention of the term PUF was by Gassend et al. in 2002. He talks about physical random functions (PRFs) and a PUF implementation using field-programmable gate arrays (FPGAs). [3]

As PUF is a function, it has inputs and outputs. However, it is not a function in a true mathematical way. It could be described as a procedure performed on a particular device. Its inputs consist of a challenge and a physical state of the device. Given the input, the PUF produces an output (called a response). Together, they form challenge-response pairs.

A hugely important property of PUFs is unclonability. It is achieved by the physical state of the device which acts as the input to the function and influences the responses produced by the PUF. The concrete details of the physical state used is what distinguishes different PUF implementations. These physical properties could be for example propagation delay in the chip circuit or bias of uninitialized memory cells to 1 or 0 state. The latter is a basis for SRAM PUF, which is a topic of this thesis. These properties are fundamentally random since they are created by uncontrollable physical processes during manufacturing. This makes them physically unclonable.

Since the physical state of the device can change with time and environment (for example temperature or input voltage variations), the challenge-response pairs can change as well. The requirement is, that for the same challenge, the responses should be similar enough for us to be able to recognize that they belong to the same challenge. The PUF responses are also required to differ from device to device even with the same challenge. [4]

Because of these properties, PUFs can be used in devices to enable secure identification, authentication as well as cryptographic key generation. More of the required properties of PUFs, their classification, possible applications and implementations will be discussed in more detail in the next sections.

1.2 PUF properties

In this section, a list of PUF properties according to [5] is described. Some of them are fundamental to all PUF constructions (the first six listed). However, not all PUFs must necessarily exhibit all of the discussed properties. In Section 2.1, properties of SRAM PUFs are discussed as they are a topic of this thesis.

Constructibility

Constructibility states, that the specific PUF instance must be 'easy' to construct. This means that the laws of physics enable such PUF implementation in the first place. The construction cost of such PUF needs to also be adequate for its application.

Evaluability

The evaluability property discusses the challenge-response mechanism of PUFs. For a specific challenge, it should be 'easy' to obtain the corresponding response. Practically this requires that the response is acquired with respect to time, space, cost and power budget of the application.

Reproducibility

Reproducibility requires that for a specific PUF instance, responses produced by the same challenge must be similar enough. The metric that measures the similarity of the responses used is the intra-Hamming distance (which will be defined later in Section 1.4.2). Practically it must be possible to recognize that the responses belong to the same challenge.

Figure 1.1 Reproducibility: responses to the same PUF challenge need to be similar [6]

Uniqueness

While reproducibility talks about the behaviour of a specific PUF, uniqueness is defined with respect to a set of different PUF instances. Given the same challenge, two PUFs must produce a response that is different enough in the given metric. The metric used is the inter-Hamming distance (which will be defined later in Section 1.4.3).

Identifiability

Identifiability is tied to both uniqueness and reproducibility. Given a single challenge, if the responses from the same PUF are similar and responses from different PUFs are different enough, the responses can be used as a means of identification of each PUF. More formal characterization of the words 'similar' and 'different' as well as a concrete protocol for PUF identification is explained in Sections 1.4 and 1.5.1.

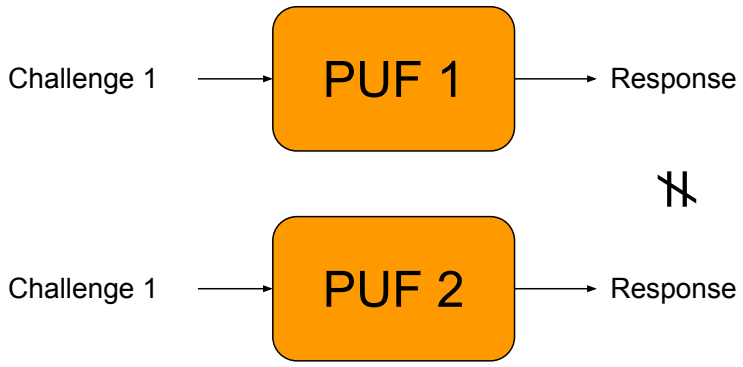


Figure 1.2 Uniqueness: two PUF instances must produce different responses if given the same challenge [6]

Physical unclonability

Physical unclonability enforces that it is hard to break the uniqueness property. The 'hardness' is expressed as a technical and physical infeasibility of creating two nearly identical PUF instances. In practice, even the manufacturer is unable to break uniqueness due to the uncontrollable physical laws on which the PUF implementation relies.

Unpredictability

A PUF is said to be unpredictable, if given a limited set of challenge-response pairs, it is impossible to create an algorithm that predicts the remaining responses based on a challenge. This means that the challenge-response pair space is sufficiently random.

Mathematical unclonability

While physical unclonability talks about the impossibility of creating a real-world clone of a specific PUF, mathematical unclonability requires that no algorithm can predict the PUF behaviour.

Mathematical unclonability is a stronger model of unpredictability. It assumes unlimited physical access to a specific PUF instance. The potential adversary can learn as many challenge-response pairs as he is capable of storing and can potentially make use of other PUF observations. Even in this situation, it should be impossible for the adversary to create a prediction algorithm capable of outputting a response based on a given challenge.

Mathematical unclonability implies that the specific PUF implementation has a large number of possible challenges (more than the adversary could theoretically store). If this was not true, a simple lookup table of challenge-response pairs could be constructed, breaking mathematical unclonability.

True unclonability

PUF is considered truly unclonable if it is physically and mathematically unclonable. True unclonability thus implies mathematical and physical unclonability.

One-Wayness

The one-wayness property is similar to the requirement of cryptographic one-way functions (for example hash functions). Given a PUF instance and a response, it should be impossible to create an inversion algorithm that can find a corresponding challenge to the response.

Similar to mathematical unclonability, the PUF needs to have a sufficiently large set of possible challenges and resulting responses in order to meet this property. Otherwise, it would be possible to construct a complete lookup table and find the wanted challenge.

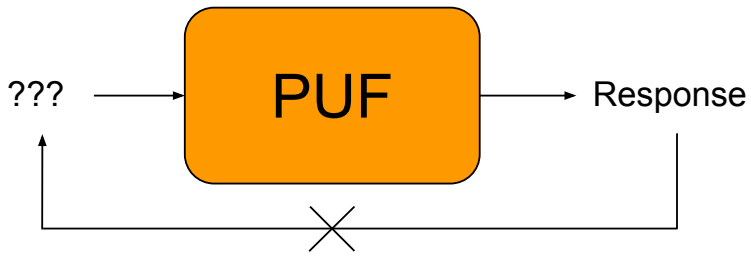


Figure 1.3 One-wayness: it is impossible to find a challenge corresponding to a given response

Tamper Evidence

Tampering with devices in order to compromise their integrity has proven to be a successful technique. For example, invasive attacks (such as microprobing) are able to extract sensitive data from the target system. [7]

Tamper evidence helps to mitigate such attack vectors. The PUF must detect the attempt to tamper with the system and change its challenge-response pairs to a different set as if the PUF instance transforms itself to a completely different one.

To achieve this property, the PUF implementation needs to rely on physical properties that will necessarily change if the device is tampered with. This change will inherently transform the PUF.

1.3 PUF classification

A lot of different PUF implementations have been proposed. They can be classified based on fabrication, security and intrinsic evaluation [8], [9]. A diagram of the discussed PUF classes is depicted in Figure 1.4.

1.3.1 Electronic, silicon and hybrid PUFs

Electronic PUFs source their randomness from electronic components. This means that they rely on properties such as capacitance, resistance or propagation delay of a circuit.

A large subclass of electronic PUFs is silicon PUFs. They can be constructed only using complementary metal–oxide–semiconductor (CMOS) technology, therefore they can be implemented on the same die together with the main chip. This prevents the need to interface with additional circuitry using external buses, lowering cost and limiting the possibility of leaking sensitive data. [1]

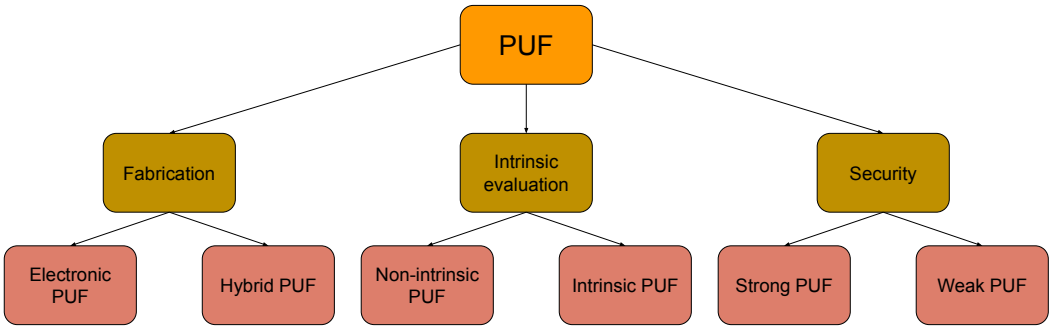


Figure 1.4 Classification of PUFs

Examples of electronic PUFs are SRAM PUF, ring oscillator physical unclonable function (ROPUF) or arbiter PUF.

Hybrid PUFs use non-electronic phenomena to create their challenge-response pairs. While randomness introduced into the system is of non-electronic nature, the signal is usually processed and stored using electronic components. Hence the name hybrid PUFs. They can be based on optics (optical PUF), magnetism (magnetic PUF) or quantum effects (quantum PUF [10]).

1.3.2 Intrinsic and non-intrinsic PUFs

A PUF implementation is said to be intrinsic if it satisfies the following two conditions [11]:

1. responses are evaluated internally
2. instance-specific random features are introduced implicitly during the manufacturing process

Internal evaluation requires the measurement equipment of the PUF to be embedded in the device. This, similar to silicon PUFs, lowers cost and is more secure.

The second condition discusses the introduction of randomness into the system. Implicit randomness relies on process variations taking place during normal manufacturing, while explicit randomness needs to be created by special procedures which would have not been needed otherwise (such as doping with random dielectric particles for the construction of a coating PUF [12]).

SRAM and arbiter PUFs are examples of intrinsic PUFs. If the PUF construction does not satisfy the given properties, it is called non-intrinsic. For example, optical or coating PUFs are non-intrinsic. [5]

1.3.3 Strong and weak PUFs

Security based classification distinguishes PUF implementations according to the size of their challenge-response pair sets.

In order for a PUF to be classified as strong, its challenge-response pair set needs to be sufficiently large to prevent an exhaustive search by a possible attacker. The challenge-response pair size thus scales well (preferably exponentially) with some construction parameter. [13]

On the other hand, weak PUFs have only a limited set of challenge-response pairs. Some PUFs can only have one pair. They are sometimes called physically obfuscated key (POK).

As the naming implies, strong PUF constructions possess, in some sense, greater security. Weak PUFs cannot be mathematically unclonable (and consequently cannot be truly unclonable

or unpredictable). Strong PUFs can also be used in more applications, as explained in Section 1.5. However, it turns out that constructing a strong PUF is a very hard problem. [1]

1.4 PUF evaluation parameters

As PUFs rely on uncontrollable physical processes to produce their responses, it is crucial to perform an analysis of their quality. Several evaluation parameters were defined by [14], [15] and [16] and they will be discussed here.

These parameters enable us to quantify the performance of PUFs and compare them. Some of them will be used to evaluate the SRAM PUF implemented in this thesis in Sections 4.1.3 and 4.2.3.

The following evaluation parameters will be defined:

- Uniformity
- Reliability
- Uniqueness
- Bit-aliasing
- Randomness

Before the parameters can be defined, a notation used in the upcoming text is described here and in Table 1.1 to avoid possible confusion.

The Hamming weight $\mathrm{HW}(x)$ of a bit string x is defined as the number of one bits in the string.

The Hamming distance $\mathrm{HD}(x, y)$ of two bit strings x and y is defined as a number of positions where the bits of x and y differ.

The reference response $R_i^{'}(n)$ is the expected response of n bits produced by the chip i. It is usually computed as a bitwise average between m response samples taken in normal operating conditions. However, some works choose the reference response as the first measured sample. [4]

Notation	Description
k	The number of devices
m	The number of response samples
n	The number of bits in a response
$R_{i,j}(n)$	The j-th response (containing n bits) of i-th chip
$R_i^{'}(n)$	The reference response of i-th chip, containing n bits
$r_{i,j}^{'}$	The j-th bit of a reference response of chip i
$\mathrm{HW}(x)$	The Hamming weight of x
$\mathrm{HD}(x, y)$	The Hamming distance between x and y

Table 1.1 Notation used in evaluation parameters definitions

1.4.1 Uniformity

Uniformity represents a proportion of zero and one states of the response bits. It is useful to determine if there exists a global bias towards any of these states. Since the PUF responses need to be unpredictable, there should not be any bias. Thus, the ideal value of uniformity is 50%.

The uniformity property for one response of chip i is defined by Equation 1.1:

$$U_i = \frac{1}{n}\sum_{j=1}^{n} r_{i,j}^{'} = \frac{\mathrm{HW}(R_i^{'}(n))}{n} \cdot 100\% \tag{1.1}$$

Uniformity can also be calculated as an average value between k PUF instances of the same type by Equation 1.2 [16]:

$$U_{\mathrm{avg}} = \frac{1}{k}\sum_{i=1}^{k} U_i = \frac{1}{k}\sum_{i=1}^{k} \frac{\mathrm{HW}(R_i^{'}(n))}{n} \cdot 100\% \tag{1.2}$$

Since uniformity only measures a global proportion of the response bit states, it is not a perfect tool to detect a possible local bias. A trivial example could be a response defined in the following way:

$$r_{i,j}^{'} = \begin{cases} 1 & \text{if } j < \frac{m}{2} \\ 0 & \text{otherwise} \end{cases} \tag{1.3}$$

That is the first half of the response bits are ones and the second half are all zero bits. Uniformity U_i for such a response is close to the ideal value of 50% though the response is clearly not unpredictable.

1.4.2 Reliability

Reliability is a parameter that measures the consistency of PUF responses to the same challenge. It is tied to the reproducibility property from Section 1.2.

In order to calculate reliability, the intra-Hamming distance for a chip i needs to be defined first [16]:

$$\text{HD-intra}_i = \frac{1}{m}\sum_{j=1}^{m} \frac{\mathrm{HD}(R_{i,j}(n), R_i^{'}(n))}{n} \cdot 100\% \tag{1.4}$$

The reference $R_i^{'}(n)$ is obtained at nominal operating conditions (room temperature and normal voltage). The responses $R_{i,j}(n)$ are then taken at different conditions and the result is calculated. It essentially represents the percentage of bits that change compared to the reference response.

Reliability is then defined by Equation 1.5:

$$\text{Reliability} = 100\% - \text{HD-intra} \tag{1.5}$$

Since the reproducibility property requires PUFs to produce similar responses to the same challenge, the ideal HD-intra_i value is 0% and thus reliability needs to be close to 100%.

1.4.3 Uniqueness

Uniqueness measures how much responses from different PUFs vary. It is estimated by the inter-Hamming distance defined by Equation 1.6:

$$\text{HD-inter} = \frac{2}{k(k-1)}\sum_{i=i}^{k-1}\sum_{j=i+1}^{k} \frac{\mathrm{HD}(R_i^{'}(n), R_j^{'}(n))}{n} \cdot 100\% \tag{1.6}$$

The original definition of HD-inter by [17] does not specify how to choose the responses to calculate the Hamming distance from. However, [4] uses the reference responses and this definition will be used in future calculations in this thesis.

The inter-Hamming distance takes all combinations of pairs of devices and computes the average Hamming distance of their responses. The uniqueness property of PUFs in Section 1.2 requires that the responses should differ as much as possible. This is achieved when HD-inter is close to its ideal value of 50%.

1.4.4 Bit-aliasing

Uniformity in Section 1.4.1 looks at the global bias of PUF responses from a single device. On the other hand, bit-aliasing measures the proportion of zero and one state of a single bit across different devices.

Bit-aliasing is calculated as the percentage Hamming weight of the j-th bit. It is defined by Equation 1.7 [16]:

$$\text{Bit-aliasing}_j = \frac{1}{k}\sum_{i=1}^{k} r'_{i,j} \cdot 100\% \tag{1.7}$$

Same as for uniformity, the ideal value for bit-aliasing is 50%. Values close to 0 or 100% would indicate bits that stay the same across different PUF instances, violating the uniqueness property.

1.4.5 Randomness

As PUF responses need to be unpredictable, they are required to contain sufficient entropy. According to [15], there are several ways to test whether the response data is sufficiently random. Two are described here, the compression test and the NIST randomness test.

NIST randomness test

The NIST randomness test uses the NIST statistical test suite to determine if the data is sufficiently random. Each test from the battery is tested for the null hypothesis that the response data is truly random on some set significance level.

Furthermore, each test is executed multiple times on different sequences of the data and a p-value for each run is calculated. These p-values are then tested for the null hypothesis that they are from a uniform distribution, as would be the case for truly random data. [18]

However, not all tests can be used as they require more input data than is practically possible to generate by the PUF. Tests for which only 170-bit string suffices are listed here [15]:

- Frequency (monobit) test
- Frequency test within a block
- Runs test
- Test for longest run of ones in block
- Serial test
- Approximate entropy test
- Cumulative sums (Cusum) test

Compression test

The idea behind compression test is simple. If a lossless compression algorithm is able to reduce the size of the tested data, it does not have full entropy[1] [15].

[1] Meaning the entropy in bits is the same as the length of the data in bits.

A good example of a compression algorithm to use is the context-tree weighting (CTW) algorithm as it was shown to produce the best results out of the commonly used entropy estimators [19].

1.5 PUF applications

Thanks to the properties described in Section 1.2, PUFs are suitable for several cryptographic applications. Three main uses of PUFs according to [5] will be described here: identification, authentication and cryptographic key generation.

1.5.1 Identification

A PUF can be compared to a device's fingerprint. The PUF provides an inherent identifying feature[2] to the entity encapsulating it. Therefore, device identification is a natural application for PUFs. A simple identification protocol using the device's PUF is described.

The process of device identification consists of two phases: enrollment and identification [5].

Enrollment phase:
In this phase, the inherent identifying feature of every considered device is collected. This means saving a PUF response from each device to a database.

Identification phase:
When the device needs to be identified, it generates its PUF response. The response is then compared to other responses stored in the database.

However, the comparison of the presented PUF response is not straightforward. Since the PUF relies on a physical state of the system, errors can arise while generating responses. For this reason, the comparison is based on the Hamming distance (HD) rather than on equality.

The property of identifiability (discussed in Section 1.2) tells us that responses from a specific device will be similar (have low HD) and responses from different devices will be far apart (have high HD). Thus, a threshold value of HD needs to be chosen. Responses with lower HD than the threshold will be considered to have originated from the same device.

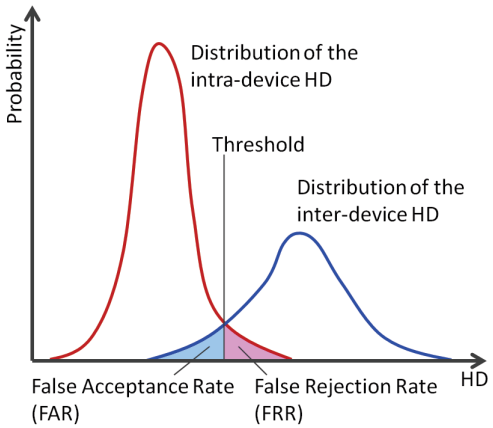


Figure 1.5 PUF identification: intra and inter HD distributions [20]

[2]Inherent identity is a unique characteristic of the device itself (a PUF response), while assigned identity is an artificially made up property (a serial number).

The threshold can be chosen based on the distributions of inter and intra HDs. While the inter-HD is a measure of how responses from different devices differ, the intra-HD indicates the similarity of responses from the same device.

If the distributions of inter and intra HDs do not overlap, the threshold value between the distributions is optimal. The false acceptance rate (FAR) (the probability of falsely identifying a device) and false rejection rate (FRR) (the probability of falsely rejecting a device) are zero.

If the distributions overlap, a compromise between FAR and FRR must be made. Reducing FAR will inevitably increase FRR and vice versa. Figure 1.5 illustrates possible intra and inter HD distributions and a chosen threshold.

1.5.2 Authentication

Authentication requires the entity, which wants to authenticate itself to a verifier, to provide proof of its identity. The entity also needs to convince the verifier that it actively participated in the creation of the proof. [5]

A simple protocol based on the challenge-response pairs of PUFs will be presented here. It is divided into two distinct phases: enrollment and authentication. [21]

Enrollment phase:

During this phase, a trusted third party (the verifier) is in possession of the device which will later be authenticated. A sufficient number of randomly generated challenges along with the corresponding responses (obtained from the device) is saved to a database.

Authentication phase:

When the device needs to be authenticated, the verifier chooses a challenge from the database and sends it to the device. The device then responds with the corresponding response. Finally, the obtained and the previously recorded responses are compared. If they are sufficiently similar, the device is authenticated successfully. This challenge-response pair is then deleted from the database and never used again.

The whole process of this PUF-based authentication protocol is illustrated in Figure 1.6.

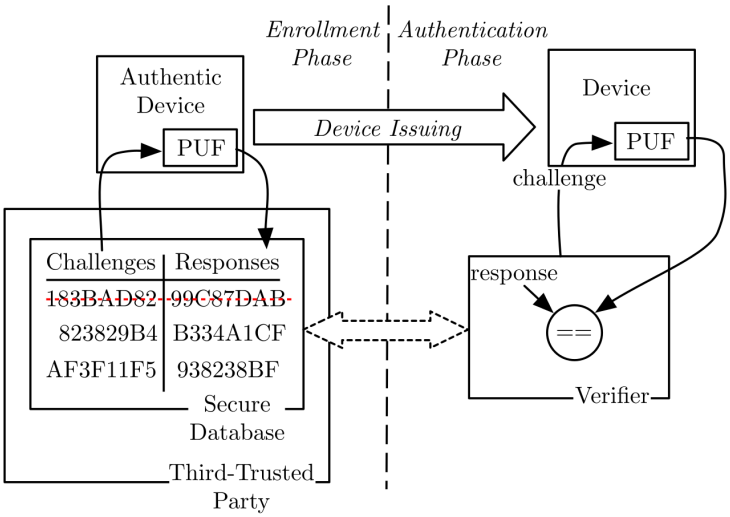


Figure 1.6 PUF-based authentication protocol [22]

The comparison of the responses is done in the same way as in PUF-based identification (Section 1.5.1). The HD is used as a distance metric and a threshold for accepting the response must be chosen appropriately.

The challenge-response pairs are deleted from the database and never reused to prevent a potential man-in-the-middle attack. Since this is not a threat, the communication between the device and the verifier does not need to be secured.

This protocol is simple and has its drawbacks. It only provides a limited number of authentications for the verifier given by the number of stored challenge-response pairs. The authentication is only one-way, the verifier is never authenticated to the device. It requires the PUF to be strong (to have a large number of available challenge-response pairs).

More advanced PUF-based authentication protocols that address some of the aforementioned drawbacks have been developed by [1], [22] and [23].

1.5.3 Key generation

Keys are required in the majority of cryptographic applications. They need to contain true randomness in order to be unpredictable and unique. The cryptographic key then needs to be stored and retrieved securely. These conditions are hard to meet and a myriad of cryptographic systems have been broken because of bad generation or handling of keys. [24]

PUF-based cryptographic key generation tries to solve those problems. The key is not stored in any memory and is generated by the PUF on demand. In some way, it is imprinted in the PUF itself. Furthermore, the uniqueness and unpredictability property of PUFs, which arise from their uncontrollable physical properties, introduce the necessary randomness.

Since the PUF responses are noisy and not 100% reliable, care needs to be taken while generating cryptographic keys. The keys need to be the same every time, otherwise the cryptographic algorithms using it would not produce the desired output. For this reason, ECCs are used to obtain the same key every time with sufficient probability.

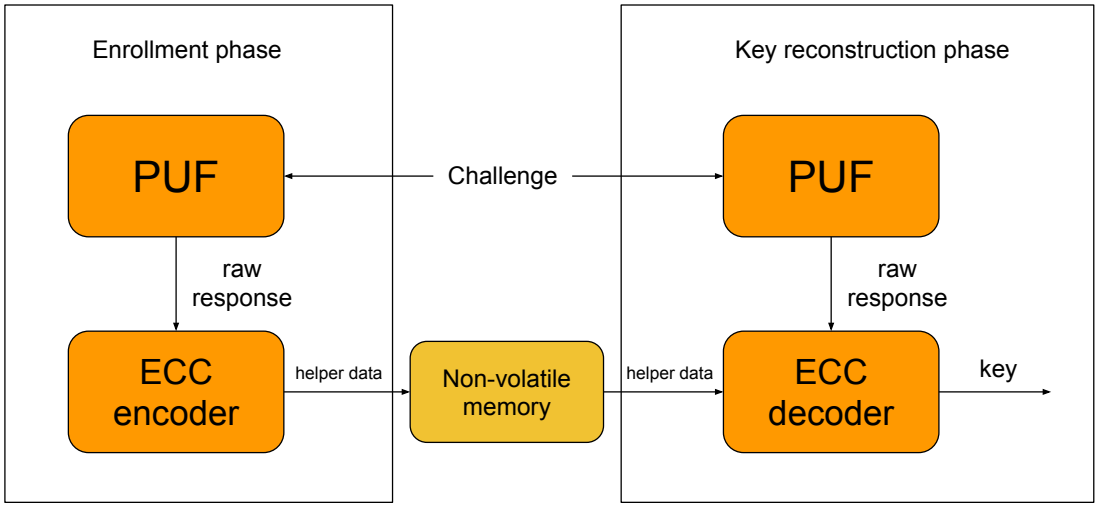


Figure 1.7 PUF-based key generation mechanism, inspired by [25]

The process of PUF-based key generation is divided into two phases: enrollment and key reconstruction [14]. It is illustrated in Figure 1.7.

Enrollment phase:
In this phase, PUF responses are generated and processed by the ECC algorithm to construct the cryptographic key. The ECC calculates helper data from the key and saves it to non-volatile memory. This data will later be used to reconstruct the key. The helper data can also contain a PUF configuration.

Key regeneration phase:
When the key is needed, the PUF produces a response and feeds it to the ECC algorithm. The key is then reconstructed by the ECC using the response and the corresponding helper data saved in the non-volatile memory.

The ECC used is usually a simple repetition code or a Bose–Chaudhuri–Hocquenghem (BCH) code. However, any ECC can be used and the codes can even be concatenated in order to increase efficiency. [26]

After key regeneration, a raw bit string key is obtained. It can be used immediately with some cryptosystems (typically symmetric ciphers such as AES). However, some systems put restrictions on the key (RSA needs two prime numbers), thus requiring further processing.

Key generation is a typical application of weak PUFs [27]. As the SRAM PUF implemented in this thesis is classified as weak, a simple key generation algorithm will be tested on top of the PUF.

It is even possible for a weak PUF to provide authentication without the need for a large number of challenge-response pairs. Once the key is generated by the weak PUF, it can be used in other cryptographic algorithms which provide authentication such as message authentication code (MAC) or digital signatures (at the cost of additional hardware/software). [27]

1.6 PUF implementations

A large number of various PUF implementations have been proposed. Four examples of these implementations have been chosen and a brief description of how they operate will be provided. This section will describe an optical PUF, an arbiter PUF, a coating PUF and a SRAM PUF. Focus will be given to the description of SRAM PUF (it will be given its own chapter), as it will be implemented in this thesis on the ESP32 microcontroller.

1.6.1 Optical PUF

A design of optical PUF was first proposed by [2] in 2001 and was called a physical one-way function.

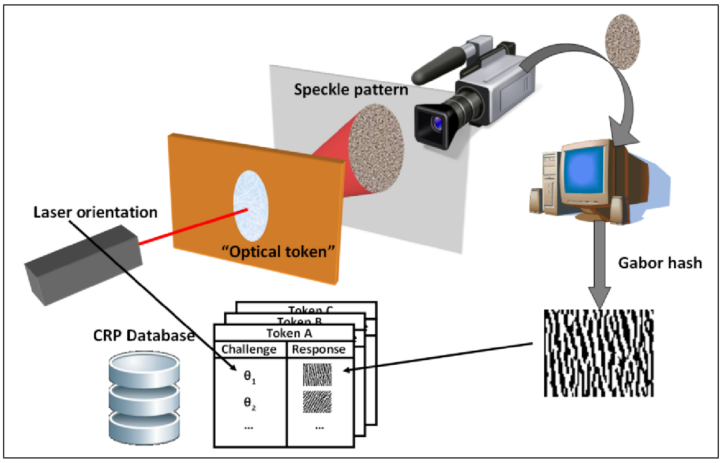


Figure 1.8 Operation of the optical PUF

The heart of the PUF is an optical token that contains microscopic refractive particles randomly mixed in a transparent epoxy plate. This optical microstructure is then irradiated with

a laser which produces a random speckle pattern. This pattern is captured using a camera and digitally processed by an algorithm called Gabor hashing. The resulting hash works as the PUF response. Challenge is represented as a specific orientation of the laser. The operation of optical PUF is represented in Figure 1.8.

Experiments showed, that this optical PUF has the one-wayness property and the optical tokens are tamper evident. [28] Together with the fact that it is classified as strong (has a large number of challenge-response pairs) makes this PUF construction the only one with these properties. However, it is also expensive and laborious to use because of its rather large and mechanical setup. [5]

A more integrated implementation of an optical PUF was proposed by [29].

1.6.2 Arbiter PUF

The arbiter PUF was first proposed by [30]. It is a delay-based silicon PUF implementation. The idea behind this construction is to create a race condition for two digital signal paths which end in an arbiter circuit. The arbiter determines which of the two digital lines finished the race first and outputs a bit.

If the two paths have an identically designed delay in the chip, the two following cases can happen [1]:

1. The delay for one of the paths is shorter due to the manufacturing variations and the arbiter circuit will decide the race accordingly. This effect is the basis of the arbiter PUF since the variations are random for every device and stay static after fabrication of the circuit.

2. By chance, the two paths have a nearly identical delay. The arbiter circuit then enters a metastable state and its output is essentially random. This randomness is however not device-specific and is the source of unreliability of the PUF.

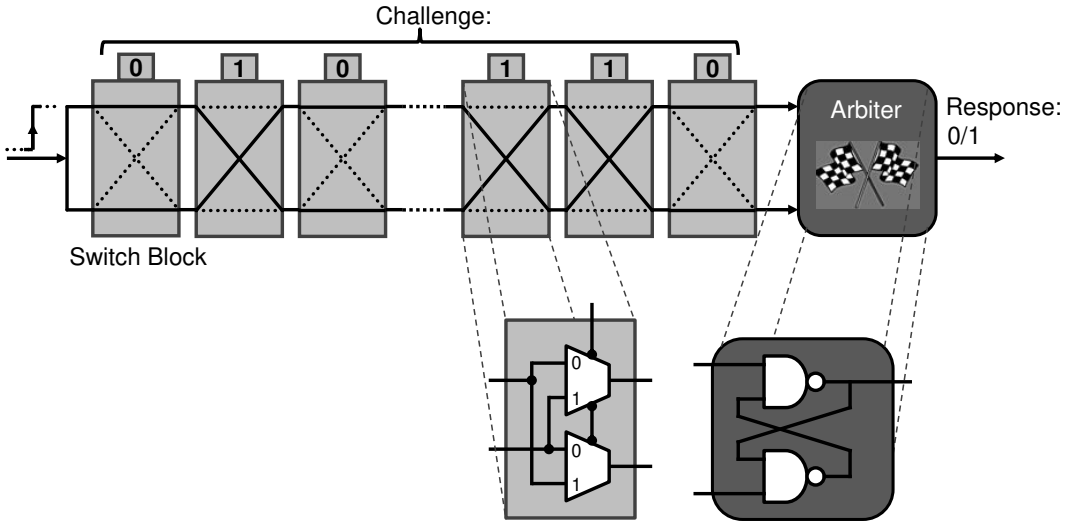


Figure 1.9 Construction of an arbiter PUF as proposed by [30], image from [5]

The initial design of the arbiter PUF by [30] used serially connected switch boxes to implement the two delay paths. A switch box connects two inputs to two output lines based on a parameter bit. The connection is either straight or switched, creating a configurable delay for the two paths.

Connecting n switch boxes results in 2^n possible delay configurations. These configuration bits are then considered to be the challenge of the PUF. The schematic of the described arbiter PUF is shown in Figure 1.9.

However, the 2^n possible challenges are not unpredictable with this design. The final delay of the path is the sum of delays on each switch box. This fact makes it possible to create a highly accurate mathematical model of the PUF while only observing a limited amount of challenge-response pairs. [30]

In order to prevent the model building attacks, extensions to the original design were proposed by introducing non-linear features. One example is a feed-forward arbiter PUF, where some of the configuration bits are set by evaluating the race at an intermediate point of the path rather than by the challenge. [31]

1.6.3 Coating PUF

The design of a coating PUF was first proposed by [32]. It gathers its randomness from measuring the capacitance by comb-shaped sensors placed in the top metal layer of an integrated circuit. The randomness is introduced into the system by spraying a passive dielectric coating on top of the sensors. The need for a special step during manufacturing to create the coating classifies this PUF construction as non-intrinsic (it breaks the second necessary condition of intrinsic PUFs as discussed in Section 1.3.2). [11]

Construction of a coating PUF is illustrated in Figure 1.10.

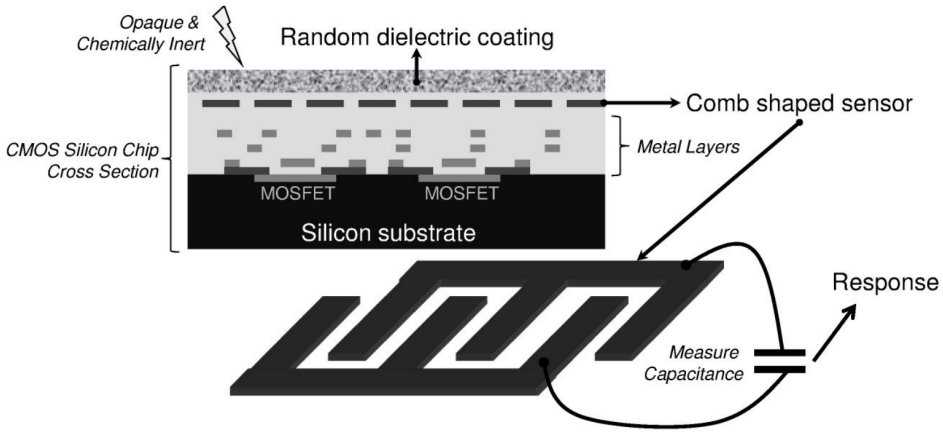


Figure 1.10 Construction of a coating PUF [5]

As the coating is inert and opaque, it provides strong protection against physical attacks. It has also been shown that the resulting PUF is tamper evident. [32]

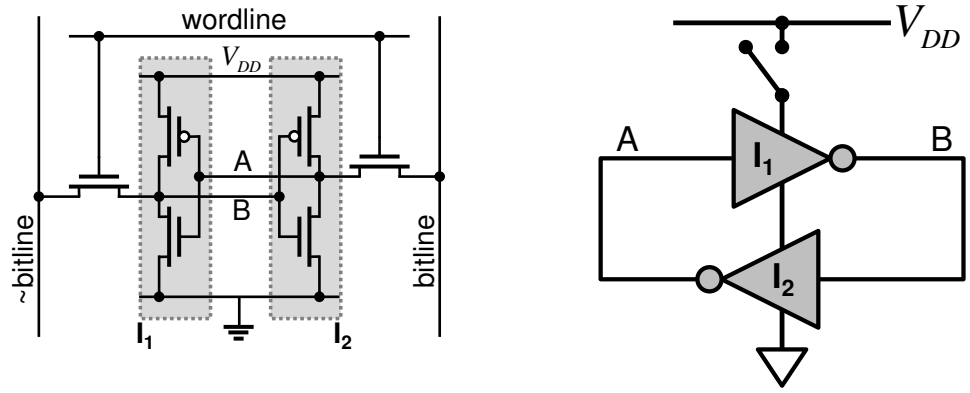
Chapter 2

SRAM PUF

SRAM PUF is based on a random startup behaviour of individual SRAM cells. It was first proposed by [13] in 2007.

In a CMOS implementation (which is used to create 99% of integrated circuits as of 2011 [33]) a SRAM cell is usually made up of two cross-coupled inverters. The inverters reinforce each other of their state, thus creating a bistable circuit—a circuit with only two stable states. This enables the cell to save exactly one logical bit. Each inverter is implemented using two transistors and additional two transistors are used to enable read and write operations. This means that storage of a single bit costs six transistors in hardware. [11]

The logical diagram of a SRAM cell containing the two inverters is illustrated in Figure 2.1b and a diagram with the transistor layout of the cell is shown in Figure 2.1a.



(a) A CMOS diagram of a SRAM cell

(b) A logic circuit diagram of a SRAM cell

Figure 2.1 CMOS and logic circuit diagrams of a SRAM cell [5]

The principle of a SRAM PUF is based on the power-up values of the memory cells. The memory cells are volatile—when power is lost, the data disappears from memory. After startup, each cell has to initialize to either a '1' or a '0' state. Which state the cell stabilizes on is dependent

on a relative strength[1] of the two inverters. Since the inverters are designed identically, their strength is determined by random process variations during manufacturing. [11]

For each cell independently, the two following cases can happen:

1. One of the inverters has far greater strength than the other. The cell has a significant preference for one initial state. Meaning it will either be in the '0' or the '1' state most of the time after power-up. This case is crucial for the PUF construction.

2. By chance, strengths of the two inverters are nearly identical. The resulting initial state depends on random noise in the circuit. This is the source of unreliability of the PUF.

During fabrication, each cell acquires its specific properties and thus its startup state stability according to the cases mentioned above. However, stability is not a discrete property. One cell can be less stable than some other. An illustration of this can be seen in Figure 2.2.

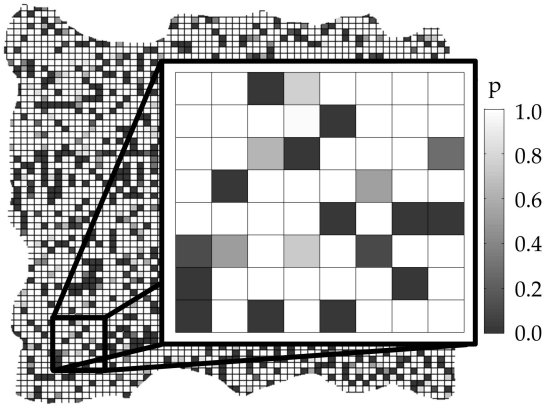


Figure 2.2 Illustration of SRAM cell stability. A small area of SRAM cells is shown. The lightness of each cell indicates the probability that the cell's startup state will be '1'. Note that most of the cells are either nearly black or nearly white. Meaning they are stable and their preferred state is '0' or '1' respectively. [34]

It turns out, that the process variations are significant enough that most SRAM cells tend to be stable. This enables the startup values of memory to be used as PUF responses. Challenge can be interpreted as a concrete address in memory. The response is then read from this address. [1]

For the construction of a SRAM PUF, stable cells are important. The unstable ones can be considered harmful as they decrease reliability. However, thanks to their unknown initial state which is determined randomly after every startup, they are useful for the construction of true random number generators (TRNGs). [34]

A huge advantage of SRAM PUFs is, that SRAM memory is usually already present in chips. If the device possesses a suitable mechanism of power state control of its memory (the ability to turn off blocks of memory, power-saving states that turn off memory), no additional hardware is needed. Therefore this type of PUF could potentially be implemented in devices that were not designed to contain a PUF at all.

Data remanence time of SRAM memory needs to be taken into account as well. The memory cell remembers its state even for a short period of time after power is lost. If power is turned on again before this time, the saved data can still be found in memory. The remanence time is greatly affected by temperature. Lower temperature results in longer retention time. This effect can be observed on the ESP32 SRAM in Sections 4.2.2 and 4.1.2.

[1] Strength here means the properties of the underlying metal–oxide–semiconductor field-effect transistor (MOSFET) transistors (such as delay).

The PUF implementation must guarantee that the memory has been unpowered long enough. Otherwise, the startup memory values used as a PUF response can be altered by a potential attacker and can be for example used to manipulate the reconstructed cryptographic key. [35]

2.1 SRAM PUF and its properties

PUFs constructions can be characterized by several properties. They have been discussed in Section 1.2. However, not every PUF needs to exhibit all of them. Therefore a discussion about what particular properties a SRAM PUF meets is provided according to [1]. The summary of properties of SRAM PUF can be found in Table 2.1.

Property	SRAM PUF
Constructibility	yes
Evaluability	yes
Reproducibility	yes
Uniqueness	yes
Identifiability	yes
Physical Unclonability	yes
Unpredictability	yes
Mathematical unclonability	no
True unclonability	no
One-Wayness	no
Tamper evidence	probably no

Table 2.1 Summarization of properties of SRAM PUF

Constructibility:
Since known implementations of SRAM PUF exist (and one is provided in this thesis), it is definitely constructible. Furthermore, the effort needed to construct the PUF is low compared to the other implementations—SRAM memory is usually already present in chips and no additional hardware is needed.

Evaluability:
SRAM PUF is evaluable because it is possible to read the startup values of the SRAM cells which are interpreted as the response. However, the effort to obtain the response can be rather high since the memory needs to be read fresh after power up (before the response is overwritten by other data). Depending on the device used and its capabilities, this could potentially require procedures that take a long time (for example rebooting the device or using a power-saving state to turn off the SRAM memory). Additional hardware could mitigate this problem, increasing manufacturing costs.

Reproducibility:
The SRAM PUF possesses the reproducibility property because memory cells have a preference for some initial state. This preference is imprinted on the cell at the time of manufacturing and does not change much afterwards (however, SRAM aging effect can alter the preference of a cell over time and this is discussed later in this section).

Uniqueness:
SRAM PUFs are unique, as each memory cell creates its preference for some initial state during manufacturing independently. For this reason, the resulting PUF responses from different devices should not be similar.

Identifiability:
As identifiability is the combination of reproducibility and uniqueness, both of which are already met, the SRAM PUF is identifiable as well. This means that the intra-HD and inter-HD distributions are sufficiently separated and the responses can be used to identify devices.

Physical unclonability:
The source of randomness of SRAM PUFs is the physical variation of individual transistors that implement the memory cells. These variations take place on a micro/nano scale and are technically infeasible to control fully. Even the manufacturer is unable to create two SRAM memory instances with similar cell startup state preferences. Thus this construction is considered physically unclonable.

Unpredictability:
Unpredictability assumes, that given a limited set of challenge-response pairs, no prediction algorithm can be designed for future challenges. Responses are interpreted as memory addresses and each cell should acquire its preference for some initial state independently. Therefore the knowledge of a challenge-response pair does not reveal any information about other pairs. For this reason, SRAM PUFs are unpredictable.

Mathematical unclonability:
Mathematical unclonability is a stronger model of unpredictability. It assumes having access to as many challenge-response pairs as can be stored. Since SRAM PUFs have only a limited number of those pairs (only one in the extreme case), a simple lookup table with all the possible pairs can be constructed easily. Thus, the property of mathematical unclonability is not met.

True unclonability:
A PUF is truly unclonable if it is physically and mathematically unclonable. Therefore, by definition, SRAM PUF is not truly unclonable since it is not mathematically unclonable.

One-wayness:
One-wayness (similar to mathematical unclonability) requires the PUF to have a sufficiently large challenge and response sets. Because SRAM PUFs are weak, they are not one-way.

Tamper evidence:
Not enough research has been conducted to declare whether SRAM PUFs are tamper evident or not [5]. However, it has been shown that it is possible to strip the device of most unrelated silicon without changing the PUF response significantly [36]. Additionally, [37] showed that it is possible to extract full PUF responses from AVR microcontrollers[2] by disabling the logic core and using a semi-invasive laser probing technique to read the startup memory values.

2.2 Stable response extraction

Since SRAM PUFs are weak, they are usually used for key generation. The resulting key must be the same every time it is reconstructed. Therefore, the stability of the response needs to be guaranteed. Several methods that increase stability exist—temporal majority voting (TMV), ECC, direct bit preselection and indirect bit preselection. [38]

[2] the experiment was conducted on an ATMega328P and an ATXMega128A1 models

Temporal majority voting

TMV samples the PUF response multiple times and the resulting response is based on a bitwise majority vote of the samples. This increases the time it takes to obtain the response significantly. Additionally, if a cell is very unstable, the majority vote for it could be very close and thus not stable.

Error correction codes

ECC correct the errors in the response, thus increasing stability. However, they need to store helper data in non-volatile memory to function, decrease response length and have computational overhead. A simple repetition ECC is used in the SRAM PUF implementation in this thesis and details will be discussed in Section 5.2.

Direct bit preselection

Stable bit preselection methods try to indicate which bits are stable and they create a mask of stable bits. This mask is saved to a non-volatile memory of the device and is later used to ignore the unstable bits during response extraction.

In direct preselection, the PUF response is first measured multiple times. Stable bits are identified based on their average startup value over multiple measurements. To identify more stable bits, measurements can be made across different operating temperatures and supply voltages. However, this is a very time-consuming technique. An example of average startup values of 1 KB of SRAM can be seen in Figure 2.3.

Figure 2.3 Average startup values of individual SRAM cells over 1000 measurements. White bits are stable '1', black bits are stable '0' and the rest are unstable bits with varying amounts of bias (according to the color bar on the right). Cells with an average value of 0.5 are the least stable. The data was obtained from ESP32 number 12 at 20°C.

Indirect bit preselection

Indirect preselection runs a test for each bit to identify its stability. For example, a stable bit can be identified by the time it takes for it to stabilize its initial state—stable bits stabilize faster than unstable ones. The test can be performed in the following way.

First, set all the bits to the '0' state. Then, turn off the memory for a short amount of time and look at which bits flipped to the '1' state the fastest. They are the stable '1' bits. Stable '0' bits can be obtained respectively. It has been shown by [39] that up to 100% stable response can be extracted by this method.

2.3 SRAM aging

Normal operation of SRAM memory unavoidably alters its physical properties. These processes can affect the stability of SRAM PUF and are called silicon aging.

The dominant effect which directly alters the stability of memory cells is called negative-bias temperature instability (NBTI). NBTI is a data-dependent aging effect. If the cell stores a '1' bit, its startup state preference slowly starts to change towards the '0' state. This is the result of a threshold voltage increase on the transistors which are switched on in the current state, whereas the switched off transistors are unaffected. This means, that the SRAM PUF has a natural tendency to become less stable over time if the startup data is left unchanged in memory. [40]

Anti-aging techniques, which try to counteract NBTI and other effects, exist. A simple example is to store an inverse of the startup data after response extraction. However, this is a security risk as leakage of this data would directly reveal the startup memory values. Another solution could be to overwrite the memory with a random string after every power up. This results in less effective anti-aging, but avoids the security risk of a data leak. [41]

2.4 Related work

Since the main objective of this thesis is to analyze the possibility of implementing a SRAM PUF on the ESP32 microcontroller, it is important to look at the already existing solutions.

The authors of [42] have performed an analysis of SRAM PUF implementations on existing IoT platforms. ESP32 was among the analyzed technologies. However, the information presented is extremely limited. The mechanism by which they obtained the PUF responses is not known and the operating conditions during the analysis were not specified. Furthermore, the analysis was performed using only two ESP32 microcontrollers. They also present uniformity and reliability evaluation of the PUF, though their results differ significantly from the findings in this thesis (presented in Chapter 4).

[43] proposed a solution for secure management of IoT devices based on blockchain non-fungible tokens. In their research, the blockchain account seed was not stored digitally but was reconstructed using a SRAM PUF. A proof of concept was developed and tested on three ESP32 development boards. They successfully reconstructed the seed using direct stable bit preselection and a repetition ECC. However, the PUF was not tested in different operating conditions and no evaluation parameters were used.

To summarize, proof of concept SRAM PUF implementations on the ESP32 microcontroller exist, though not enough information about how the implementations work has been provided. To the best of our knowledge, an analysis of performance of SRAM PUFs on ESP32 in different operating conditions has not been published yet.

Chapter 3
ESP32 platform

In this chapter, a brief explanation of the ESP32 platform is presented. The main hardware and software features of the ESP32 microcontroller, which is used in this thesis, are described. The information provided is mainly based on the official ESP-IDF (the official development framework) [44] and the ESP32 Technical reference manual [45].

3.1 Hardware

ESP32 is a system on a chip (SoC) microcontroller designed by Espressif which integrates CPUs, WiFi, Bluetooth and various specialized co-processors. The main features of the ESP32 are the following:

- Xtensa dual-core 32-bit LX6 microprocessor running at 160 or 260 MHz
- 520 KB SRAM, 448 KB ROM
- 8 KB FAST RTC SRAM and 8 KB SLOW RTC SRAM
- Wi-Fi: 802.11 b/g/n
- Bluetooth: v4.2 and BLE
- Cryptographic co-processors: RNG, RSA, SHA, AES
- Ultra low power (ULP) co-processor

One possible form factor of the ESP32 microcontroller can be seen in Figure 3.1. A block diagram of ESP32 functions is illustrated in Figure 3.2.

Figure 3.1 The ESP32-WROOM-32U module back and front image [46]

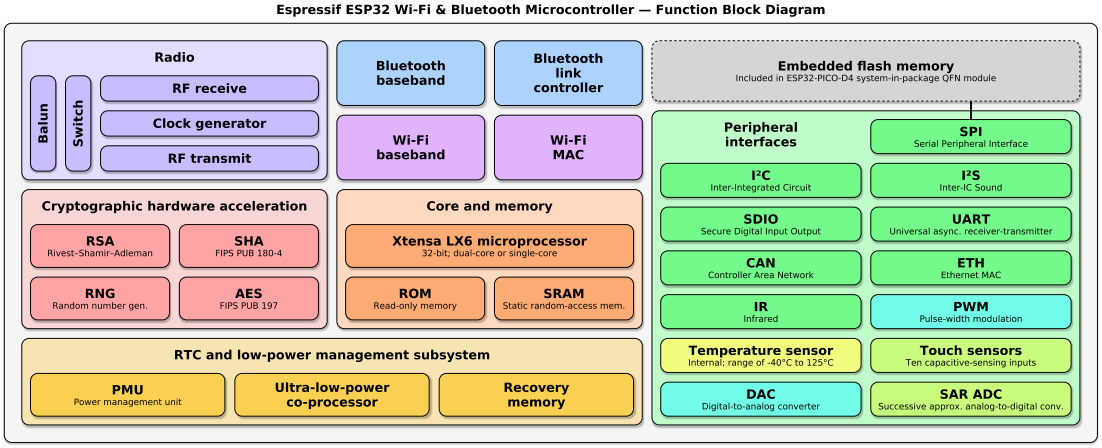


Figure 3.2 ESP32 chip function block diagram [47]

ESP32 is a name for the original microcontroller from Espressif released in 2016. However, several other models have been developed since [44]:

ESP32-S2 A single-core version of the ESP32

ESP32-S3 Version with added instructions to accelerate machine learning

ESP32-C3 Single-core RISV-V CPU version instead of the Xtensa LX6 CPU

ESP32-S6 Single-core RISC-V CPU with WiFi 6 support

Only the original ESP32 microcontroller was used during experiments and implementation in this thesis. However, the possibility of implementing a SRAM PUF on the other versions is discussed later in Chapter 4.

3.2 Memory layout

Memory layout of the ESP32 is important for a SRAM PUF implementation as the exact memory region used to extract the response needs to be considered. The ESP32 has several types of embedded memories and they are discussed here briefly. A detailed description of embedded memory address mapping can be seen in Table 3.1. The ESP32 uses a Harvard memory architecture. This means, that instructions and data use separate memories with dedicated buses. [45]

ROM 0 and 1
Read only memory (ROM) is used by the hard-coded first-stage bootloader for instructions (ROM 0) and static data (ROM 1).

SRAM 0, 1 and 2
SRAM 0 is used for program instruction. SRAM 1 is mapped to both instruction and data bus simultaneously. It can also be mapped over part of ROM 0. SRAM 2 is used for program data.

FAST RTC SRAM The 8 KB FAST real-time clock (RTC) SRAM is mapped to both instruction and data buses and is not turned off during power-saving modes (such as deep or light sleep). It is therefore used to keep data between the power-saving modes. Only the PRO_CPU[1] can use this memory.

[1] The two CPU cores are named PRO_CPU and APP_CPU

SLOW RTC SRAM The 8 KB SLOW RTC SRAM is mapped to a region shared by both the instruction and data buses. It is clocked slower compared to the other SRAM memories. It is not turned off during power-saving modes and can be used to keep data between them. The ULP co-processor also uses it as its memory.

Flash memory can either be embedded inside the SoC or connected externally using Serial Peripheral Interface (SPI). A maximum of 16 MB of flash is supported and up to 8 MB of external SRAM can also be connected using SPI.

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF8_0000	0x3FF8_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
	0x3FF8_2000	0x3FF8_FFFF	56 KB	Reserved	-
Data	0x3FF9_0000	0x3FF9_FFFF	64 KB	Internal ROM 1	-
	0x3FFA_0000	0x3FFA_DFFF	56 KB	Reserved	-
Data	0x3FFA_E000	0x3FFD_FFFF	200 KB	Internal SRAM 2	DMA
Data	0x3FFE_0000	0x3FFF_FFFF	128 KB	Internal SRAM 1	DMA
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Instruction	0x4000_0000	0x4000_7FFF	32 KB	Internal ROM 0	Remap
Instruction	0x4000_8000	0x4005_FFFF	352 KB	Internal ROM 0	-
	0x4006_0000	0x4006_FFFF	64 KB	Reserved	-
Instruction	0x4007_0000	0x4007_FFFF	64 KB	Internal SRAM 0	Cache
Instruction	0x4008_0000	0x4009_FFFF	128 KB	Internal SRAM 0	-
Instruction	0x400A_0000	0x400A_FFFF	64 KB	Internal SRAM 1	-
Instruction	0x400B_0000	0x400B_7FFF	32 KB	Internal SRAM 1	Remap
Instruction	0x400B_8000	0x400B_FFFF	32 KB	Internal SRAM 1	-
Instruction	0x400C_0000	0x400C_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data Instruction	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory	-

Table 3.1 ESP32 embedded memory address mapping [45]

3.3 Device startup

There are several steps involved between power-up and the start of the user application. It is important to understand them as some will be crucial to the implementation of the SRAM PUF. The steps are following:

1. The CPU executes the first-stage bootloader located in ROM. If the device was reset from deep sleep, a *deep sleep wake stub* is executed. The second-stage bootloader is then loaded from the flash and executed.

2. The second-stage bootloader loads the partition table from flash. The rest of the application is then loaded according to the partition table. Secure boot, over-the-air updates and flash encryption are also implemented in the second-stage bootloader.

3. C runtime and FreeRTOS are initialized. Finally, the main task of the user application is called and the booting process is complete.

The *deep sleep wake stub* is a mechanism that enables the user to run custom code early in the boot process after waking up from deep sleep. It is implemented as a function stored in the FAST RTC SRAM. This function is then called by the first-stage bootloader before it loads the second stage. [44]

3.4 ESP-IDF

ESP-IDF is an integrated development framework by Espressif. It contains libraries and source code for the ESP32 family of microcontrollers and provides a unified application programming interface (API) which is shared between the different microcontroller models. It also provides scripts for operating the toolchain, uploading applications and monitoring them. [44]

The latest stable version of ESP-IDF (on May 2, 2022) is 4.4 and this version was used for all code used in this thesis on the ESP32 microcontrollers.

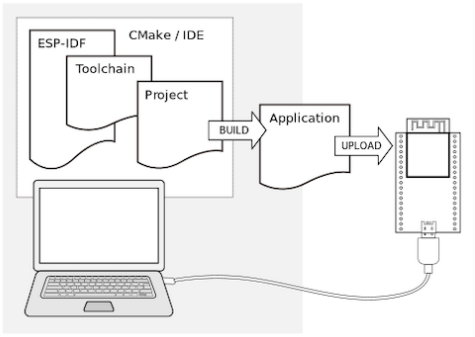


Figure 3.3 Development of applications for ESP32. [44]

3.5 FreeRTOS

A modified version of the real-time operating system (RTOS) FreeRTOS is used in ESP-IDF to create applications. Its main feature is enabling symmetric multiprocessing in order to utilize both CPU cores. Tasks play the role of threads and Wifi, Bluetooth and user tasks can be created and are managed by a real-time scheduler. The FreeRTOS also provides inter-task communication and synchronization API. [44]

Chapter 4

SRAM PUF implementation on ESP32

The goal of this chapter is to find a suitable mechanism for SRAM power state control on the ESP32 microcontroller. Two different methods are presented and their result (the startup memory values used as a PUF response) are analyzed.

First, an analysis based on temperature and SRAM power off-time is performed. The temperature is controlled by a temperature chamber Binder MK-56obj. It is capable of producing temperatures from -40 °C to +180 °C [48]. However, only the range of -40 °C to +70 °C is used for the analysis as higher temperatures could potentially damage the components. This analysis is performed mainly to understand the data retention time of the SRAM and set an adequate minimum interval the memory needs to stay off before extracting the response.

Then, some of the PUF parameters defined in Section 1.4 are used to evaluate the results. A summary of both power state control methods is provided at the end, discussing their advantages, disadvantages and viability of use.

Only the original ESP32 microcontroller model is used. Sixteen pieces will be measured and they are numbered from microcontroller unit (MCU) 1 to MCU 16. All of the measurements were captured under normal voltage conditions.

4.1 RTC SRAM based PUF

The first method of SRAM power control is to use the RTC part of SRAM memory on ESP32. This part of memory can be switched on and off using the RTC power domain controller.

4.1.1 RTC SRAM power control

The ESP32 has the ability to power-control parts of its chip. This feature can be used to save power while some of the components of the microcontroller are not being used. WiFi, digital core, internal SRAM and RTC SRAM can all be power-controlled. This seems like an ideal mechanism for implementing the SRAM PUF as the memory can be turned off and on during runtime with little time overhead. [45]

All of the individual SRAM sections (SRAM 0, 1 and 2, RTC FAST SRAM and RTC SLOW SRAM, described in more detail in Section 3.2) can be power-controlled individually. One of the sections, therefore, needs to be selected for the PUF implementation.

At first, the logical SRAM section to select would be the SRAM 0, 1 or 2, as they have a large capacity (all three more than 100 KB). However, all of the data saved in the selected section

would be lost during response extraction (the memory needs to be turned off). This would prohibit this chunk of memory from being used and potentially render some memory-hungry applications impossible to be implemented. Additionally, the ESP-IDF toolchain would need to be adjusted to not use this part of memory.

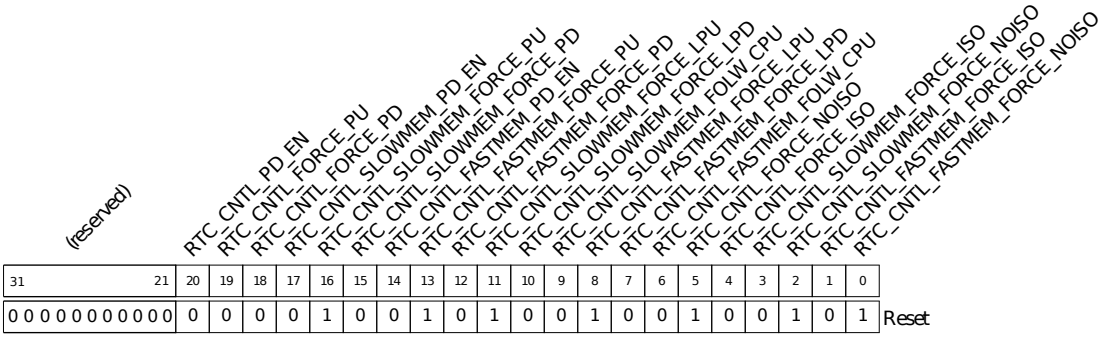


Figure 4.1 RTC domain power management register (address 0x3FF48080) [45]

Because of these drawbacks, the RTC FAST SRAM section was selected. Since it is also used to store program instructions (the deep sleep wake stub) and data (dynamic heap allocations), it needs to be preserved during response extraction. This can be done by backing up the whole 8 KB section of memory to the main SRAM before powering down and restoring it afterwards. This is not possible with the main SRAM sections because of their size.

The power state of the FAST RTC SRAM is controlled by the RTC domain power management register. Functions of the individual register bits are shown in Figure 4.1. Bits number 13 (`RTC_CNTL_FASTMEM_FORCE_PU`[1]) and 12 (`RTC_CNTL_FASTMEM_FORCE_PD`[2]) are important. They can be set to control the power state of the memory. [45]

4.1.2 SRAM analysis based on temperature and power-off time

Data remanence of the RTC FAST SRAM, power controlled using the above-described method, is analyzed in this section. Several experiments for each MCU have been conducted according to this algorithm:

1. Set the ambient temperature to the desired value.
2. Initialize memory to a default value.[3]
3. Turn memory off for a set interval and then turned it on again.
4. Observe the contents of memory.
5. Continue from step 2. with an increased turn off interval until enough data is collected.

Observing the contents of memory, in this case, means calculating the percentage of bits that flipped to a different state while turned off. For the default initial state of '0', this is equivalent to calculating the percentage Hamming weight of the bit string (counting the '1' bits).

[1] `RTC_CNTL_FASTMEM_FORCE_PD` = RTC control fast memory force power up
[2] `RTC_CNTL_FASTMEM_FORCE_PD` = RTC control fast memory force power down
[3] In the presented experiments, the default value was '0' for all bits. However, measurements with bits initialized to '1' showed nearly identical results.

The expected result is that, at the beginning for a very small turn off interval, no bits will flip. The biggest interval under which no bits flip is the *data remanence time*. With increasing turn off interval, more and more bits start to flip. The percentage of flipped bits should stabilize around 50% which is the ideal value of uniformity. At this point, all data previously saved in memory should be lost and the contents of memory after power up should depend only on the preferred initial state of the memory cells. The smallest interval under which this happens will be called the *fade-out time*. The SRAM PUF implementation should set the turn off interval to a greater value than the maximum *fade-out time* of the SRAM under varying conditions (such as temperature or voltage). This prevents data remanence from affecting the PUF response.

The results of the first experiment are displayed in Figure 4.2. The graph shows the percentage of flipped bits at 20 °C with the turn off interval ranging from 0 to 1000 μS. The results are as expected. The *data remanence time* is around 180 μS, the *fade-out time* is around 900 μS and the percentage of flipped bits stabilizes around 50% for all MCUs.

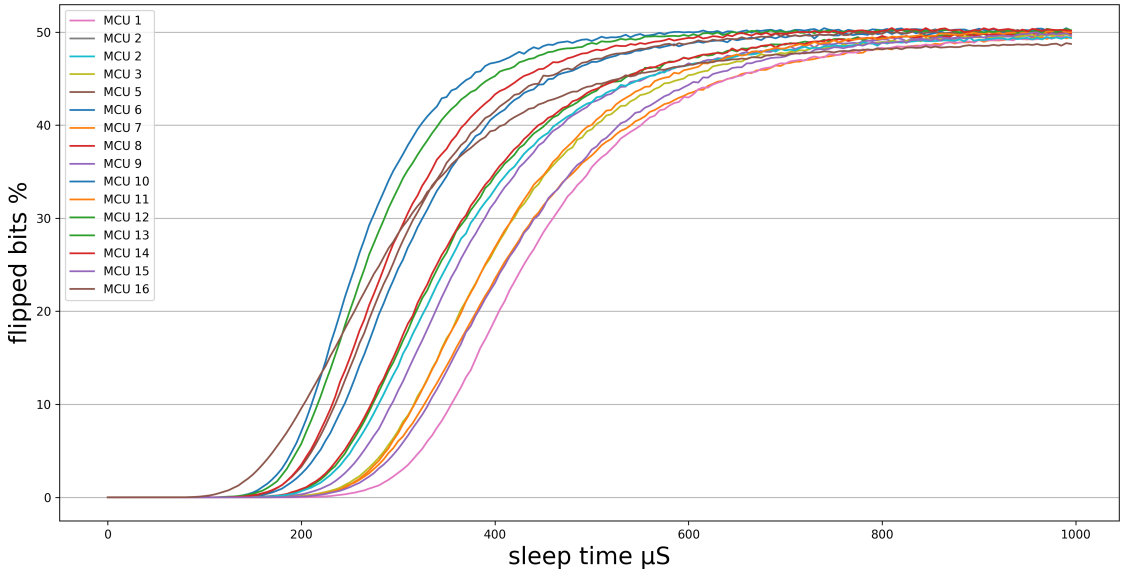


Figure 4.2 Percentage of flipped bits of all MCUs at 20 °C (RTC SRAM method)

The second experiment shows the behavior of one MCU (number 6 was chosen but results for the others are similar) across the whole range of -40 to +70 °C. The graph can be seen in Figure 4.3. As expected, the lower temperatures increase the *data remanence time* and the *fade-out time* significantly and higher temperatures decrease them. However, for the lower temperatures (below 0 °C), the *fade-out time* cannot be seen as the memory did not have enough time to stabilize yet.

The next experiment, therefore, looks at the behavior of all MCUs at -20 °C. The turn off interval was significantly increased to 50 000 μS. The results are displayed in Figure 4.4. A very strange behavior can be seen. It looks as if the memory does not fade-out at all, only a limited number of bits flip and the rest of the bits retain their original state. Furthermore, for each board, a wildly different number of bits flip in this temperature. Nearly no bits flip for MCU 4, but about 33 percent of bits flip for MCU 6.[4]

This was confirmed by extending the range of the turn off interval to 1 S and then to 100 S. The number of flipped bits does not change anymore with increased turn off time and the memory 'freezes'. At -40 °C only about 1 % of bits flip no matter the turn off interval for each MCU. The initial value of the memory cell does not matter either, '0' and '1' states both freeze in the same way.

[4]Note the values on the Y-axis of the graph, they do not even reach 50 %.

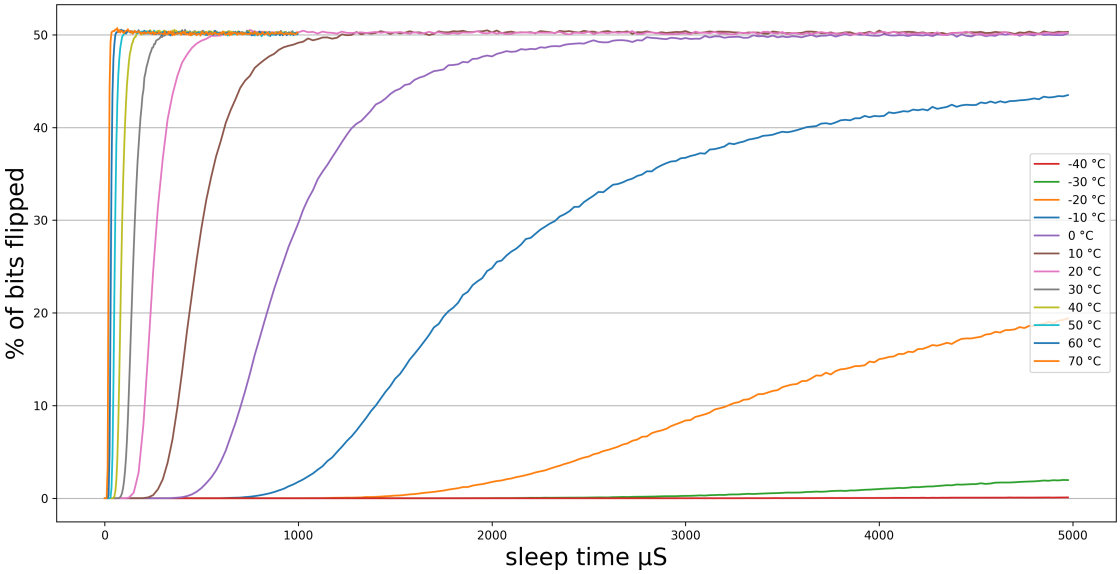


Figure 4.3 Percentage of flipped bits of MCU 6 across a range of temperatures (RTC SRAM method)

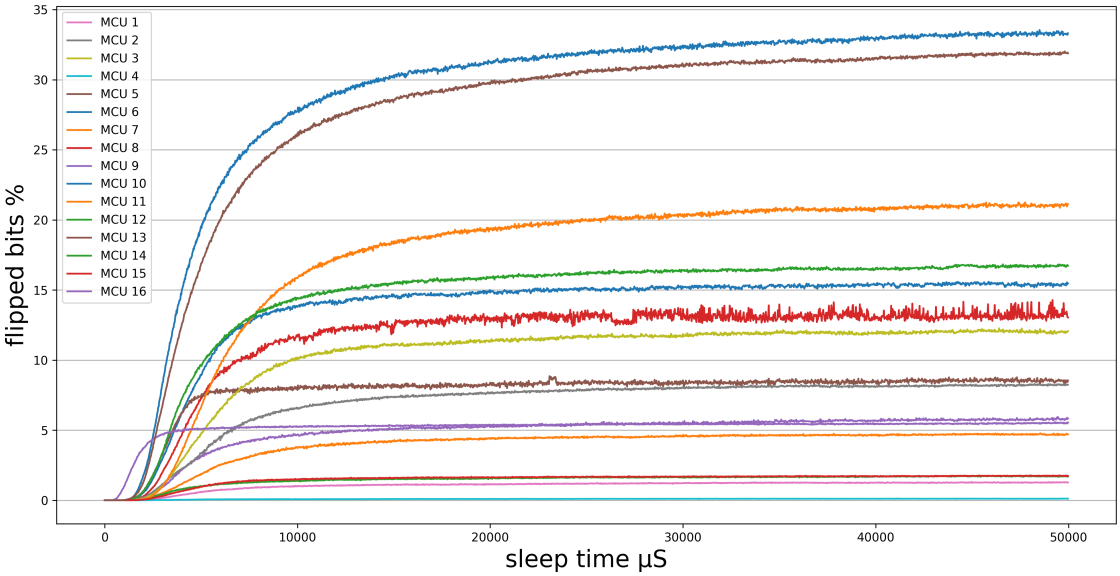


Figure 4.4 Percentage of flipped bits of all MCUs at -20 °C (RTC SRAM method)

A possible explanation of this behavior can be the following. All transistors leak current even if they are switched off. This is called a leakage current. A transistor that works as a switch for turning the RTC FAST SRAM on and off has a leakage current that is high enough that the memory is able to retain its state in low temperatures. SRAM memory draws very little power while only retaining data—no read/write operations are taking place and thus the logical states of the circuit do not change. This is called static CMOS power dissipation. It has been shown by [49] that the static power dissipation is highly influenced by temperature. A change of temperature from 100 to 20 °C decreases the power draw by a factor of 3 to 83, depending on the exact CMOS technology used.

As is evident from the Figure 4.4, different MCUs freeze at different rates—the percentage

of bit flips varies wildly across MCUs. This supports this hypothesis as the leakage current of the transistor is probably also influenced by the random variations during the manufacturing process.

This can explain the 'freezing' effect of memory in low temperatures. However, this explanation could not be confirmed.

4.1.3 PUF evaluation parameters

Since the evaluation parameters require PUF responses for their calculation, they need to be obtained. The response was chosen to be a 4 KB bit string of the (first half of) RTC FAST SRAM. Only one response is extracted, no memory address based challenge system was considered because of the limited size of the used memory region (8 KB). The turn off interval was set to 10 000 μS. This is a lot higher value than the *fade-out time* of the memory for warmer temperatures (above 0 °C). There is no point in considering lower temperatures since the memory freezes while using this power control technique. The reference response was calculated using a bitwise average of 1000 measured responses.

Uniformity

Uniformity of all MCUs in different temperatures can be seen in Table 4.1. The values look very good at temperatures above 10 °C, hovering around the desired value of 50 %. However, in lower temperatures, the effect of memory freezing can be seen clearly. The values plummet and hit nearly 0 % at -40 °C. This is due to the choice of default initial value of '0' bits. Uniformity rises towards 100 % if the default value is a '1' bit.

MCU	-40°C	-30°C	-20°C	-10°C	0°C	10°C	20°C	30°C	50°C	70°C
1	0.00	0.06	1.26	10.27	38.00	48.94	49.93	49.89	49.92	49.91
2	0.05	0.55	8.32	25.05	44.08	49.29	49.52	49.65	49.81	49.88
3	0.23	1.88	12.06	34.63	46.89	49.76	49.92	49.91	49.98	49.99
4	0.00	0.00	0.10	2.74	14.74	41.72	49.95	50.18	50.36	50.38
5	0.16	1.85	8.09	27.40	44.46	49.96	50.02	50.06	50.11	50.12
6	1.11	10.82	33.17	47.60	50.11	50.18	50.09	50.08	50.06	50.04
7	0.45	7.71	21.45	39.26	49.45	50.11	50.17	50.20	50.29	50.29
8	0.18	2.41	14.03	25.69	45.20	49.96	49.84	49.92	49.98	50.00
9	0.07	0.57	5.69	20.71	39.98	49.14	49.76	49.79	49.78	49.82
10	0.17	2.91	15.35	37.38	48.65	49.83	49.96	49.99	50.03	50.04
11	0.05	0.40	4.75	23.70	43.60	49.88	50.24	50.18	50.26	50.31
12	0.00	0.07	1.63	16.19	38.21	48.25	49.96	49.96	49.89	49.86
13	0.80	12.58	31.70	44.66	49.76	49.98	50.08	50.10	50.07	50.01
14	0.18	2.44	16.75	39.50	49.43	50.09	50.16	50.19	50.14	50.17
15	0.01	0.11	1.74	17.96	36.05	48.30	50.13	50.18	50.17	50.13
16	0.77	2.02	5.48	13.92	30.52	44.13	49.55	49.74	49.81	49.90
mean	0.27	2.90	11.35	26.67	41.82	48.72	49.96	50.00	50.04	50.05

Table 4.1 Uniformity (%) across all MCUs and temperatures (RTC SRAM method)

Reliability

Reliability values of all MCUs in different temperatures are displayed in Table 4.2. The reference response used for the calculations was obtained at 20°C.

Once again, the values look good in temperatures above 10 °C, being fairly close to the ideal 100 %. The highest reliability can be observed at 20 °C. This is the result of the reference response being calculated at this temperature. The effect of freezing is once again evident in low temperatures. Reliability goes down toward 50 %, which is the worst possible value.[5]

MCU	-40°C	-30°C	-20°C	-10°C	0°C	10°C	20°C	30°C	50°C	70°C
1	50.16	52.80	63.34	80.56	87.36	93.70	96.55	95.41	93.96	93.25
2	49.71	50.06	54.19	71.71	88.36	94.19	97.35	95.71	93.72	93.06
3	50.04	50.10	51.61	65.27	84.11	89.41	96.99	93.54	90.39	89.82
4	50.46	52.60	62.97	71.01	81.68	91.70	96.65	95.61	94.22	93.65
5	50.33	50.82	55.72	68.29	80.18	90.17	96.75	94.94	93.05	92.53
6	50.20	51.86	57.55	71.77	81.84	93.09	96.63	95.67	94.48	93.81
7	50.08	50.13	51.27	59.75	82.78	90.01	96.86	94.07	89.99	88.93
8	50.56	51.04	58.27	72.19	83.89	92.07	96.70	94.55	91.32	90.30
9	50.29	51.91	60.69	79.92	87.22	93.40	96.75	95.59	93.94	93.16
10	50.09	50.09	50.19	52.76	64.25	88.26	97.45	93.60	88.17	86.39
11	50.91	59.56	77.59	86.35	91.16	94.87	96.38	95.92	95.14	94.30
12	50.31	57.04	68.94	78.72	86.82	93.98	96.53	95.81	94.73	94.08
13	50.69	61.81	77.67	81.53	87.42	95.28	96.65	96.06	95.22	94.47
14	50.00	52.19	65.25	82.92	89.61	94.80	96.61	95.70	94.50	93.70
15	49.89	49.99	51.60	66.71	82.06	89.75	96.94	93.16	89.77	89.06
16	51.17	52.33	55.64	63.53	78.56	90.84	98.31	92.24	80.72	76.13
mean	50.31	52.77	60.15	72.06	83.58	92.22	96.88	94.85	92.08	91.04

Table 4.2 Reliability (%) across all MCUs and temperatures (RTC SRAM method)

Uniqueness

Uniqueness of the responses from different MCUs across the whole range of temperatures is shown in Table 4.3. Again, the values look good in temperatures above 0 °C, attacking the ideal of 50 %. However, in lower temperatures uniqueness plummets rapidly as the responses start to freeze. Since the initial value of memory is the same across the MCUs, the data remanence kicks in and the responses start to become more and more alike.

4.1.4 Summary

As is evident from the experiments, this power control method can only be used at temperatures above 10 °C. The exact temperature where the memory starts to freeze also differs for each MCU.

Moreover, only the original ESP32 and the ESP32-S2 (the single-core version) microcontrollers have the functionality to power-control the memory sections. In the newer models, this functionality was replaced by the 'retention' mode of memory. This mode enables the memory to retain its data while no read/write operations need to take place, saving power in the process. [45] Thus, this method cannot be applied to the newer models. Additionally, only 8 KB of memory is available. Though this is enough if the PUF is used for key generation, as will become evident in Chapter 5.

On the other side, the ability to power-control the memory during runtime is very powerful. The response can be extracted on demand quickly because the whole device does not need to be powered down and then up which is an extremely time-consuming process.

[5] The value of 0 % would mean that the response bits are just inverted compared to the reference.

Temperature °C	Uniqueness %
-40	0.53
-30	5.63
-20	20.18
-10	39.19
0	48.64
10	49.81
20	49.73
30	49.66
40	49.60
50	49.55
60	49.54
70	49.54
mean	38.47

Table 4.3 Uniqueness (%) across all MCUs and temperatures (RTC SRAM method)

4.2 Deep sleep based PUF

The second method of SRAM power control is to use the internal SRAM memory on ESP32. This part of memory can be switched on and off using the deep sleep mode.

4.2.1 Deep sleep SRAM power control

During the deep sleep power-saving mode, most of the chip is powered down. The CPUs, most of SRAM and all digital peripherals are turned off. Only the RTC controller, the ULP coprocessor and the FAST and SLOW RTC SRAM remain powered on.

Several sources can be used to wake the device up from deep sleep. The RTC timer, external general-purpose input/output (GPIO), touch sensor and the ULP coprocessor can all be used for this purpose. Since most of the device was turned off, the whole booting process (described in Section 3.3) must take place during wakeup.

The RTC timer is the wake-up mechanism that was chosen for the SRAM power control, as it enables setting a precise power off interval. However, the RTC timer uses an internal 150kHz RC oscillator as its clock source. The exact oscillator frequency is affected by temperature. This results in a time drift for different operating temperatures. Fortunately, this should not affect the PUF implementation. The same time drift will occur both in the real response extraction and during experiments. Only the power off timing information has reduced accuracy. This was not an issue while using the RTC SRAM based power control method, since the CPU (which performed the timing) is clocked using a 480 Mhz phase-locked loop (PLL) clock. [45], [44]

Care needs to be taken while extracting the startup memory values. As the boot process of the device is fairly complex, it is difficult to know exactly what regions of memory stay uninitialized and what regions are used by the bootloaders or the FreeRTOS before our code is executed. Even if such an uninitialized region of memory was found, it is not guaranteed that it will stay this way after a software update. The linker can assign new variables to this memory region or the old variables might be reordered. The memory would then be set to program data before we could extract the startup values.

To prevent this from happening, a process for extracting the startup values early in device boot was designed using the *deep sleep wake stub*:

1. After wakeup, the first stage bootloader calls the custom *deep sleep wake stub* function

2. The *deep sleep wake stub* runs before the majority of other boot code. Therefore, it sees nearly the whole SRAM memory uninitialized. A 4 KB of data from address 0x3FFB0000 is copied to a statically allocated buffer.
3. The rest of the boot process continues.
4. After boot, the startup data is found untouched in the buffer.

The 0x3FFB0000 address (which is an address into the internal SRAM 2) was chosen arbitrarily. Any address of the internal SRAM 0, 1 or 2 can be chosen except for their first few KB—they are used by the first-stage bootloader before the *deep sleep wake stub* runs.

The statically allocated buffer is not overwritten during the boot process because it is annotated by a *NOINIT_ATTR* attribute in the source code and is put into the *no-init* section by the linker. This section is not initialized and was created to keep data during software resets. [44]

Copying the data to this buffer by the *deep sleep wake stub* is necessary as the *no-init* section can be moved by the linker to a different memory address. This could be prevented by altering the linker script in the ESP-IDF source. Though this would make the deployment of the resulting PUF implementation significantly more difficult.

4.2.2 SRAM analysis based on temperature and power-off time

All of the experiments shown in this section were conducted using the same method that was explained at the beginning of Section 4.1.2. The above-described deep sleep power-control mechanism was used and the number of bit flips was calculated from 4 KB memory samples.

The first experiment shows the percentage of flipped bits across all MCUs at -40 °C. The lowest temperature is shown as it is the most indicative of what value to choose for the turn off interval. The resulting graph can be seen in Figure 4.5. The *data remanence time* for the MCUs is approximately 1 200 μS (this is more evident if the graph is zoomed in) and the *fade-out time* is around 50 000 μS.

The effect of memory freezing in low temperatures is not present while using the deep sleep power-control method, as all the MCUs stabilize around 50 % of flipped bits.

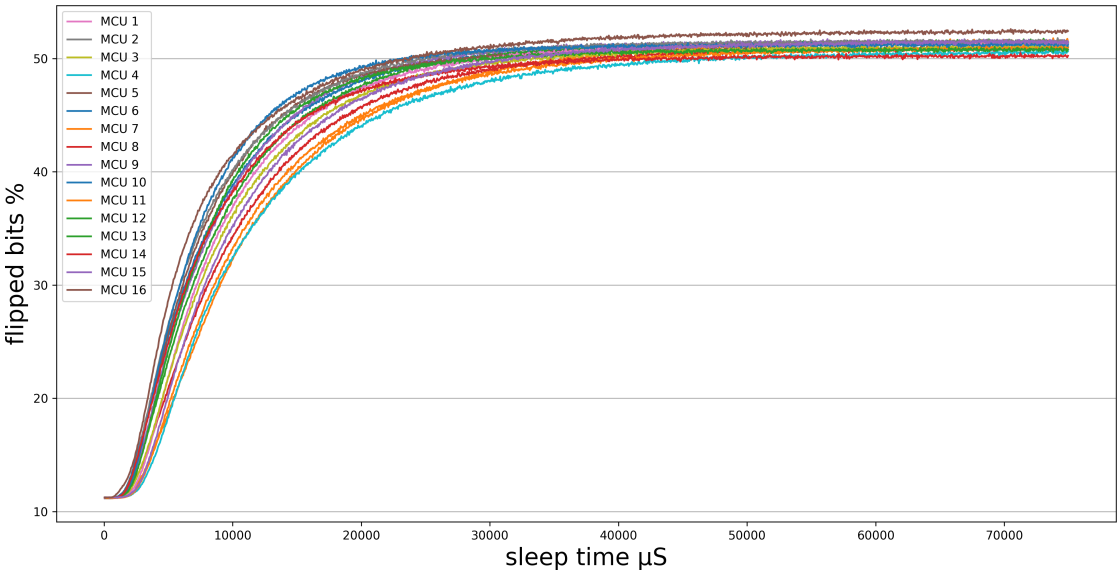


Figure 4.5 Percentage of flipped bits of all MCUs at -40 °C (deep sleep method)

The second experiment looks at the behavior of one MCU (again, the number 6 was chosen but results for the others are similar) across the whole range of -40 to +70 °C. The results are shown in Figure 4.6. This time, since no memory freezing effect is present, the memory stabilizes around the value of 50 % of flipped bits for all temperatures. It is also evident that the *fade-out time* and the *data remanence time* increase significantly with decreasing temperatures.

Note the data points for temperatures above 0 °C. Unexpectedly, they show a significant number of bit flips for very small turn off intervals (even for 0 μS). This behavior is caused by the RTC controller that wakes up the chip from deep sleep. Even if the sleep time is set to 0 μS, the whole chip is put into deep sleep for at least 1 000 μS. This is probably caused by the design of the RTC controller or the internal ESP-IDF software components. However, the deep sleep mode behaves as expected if the turn off interval is greater than 1 000 μS.

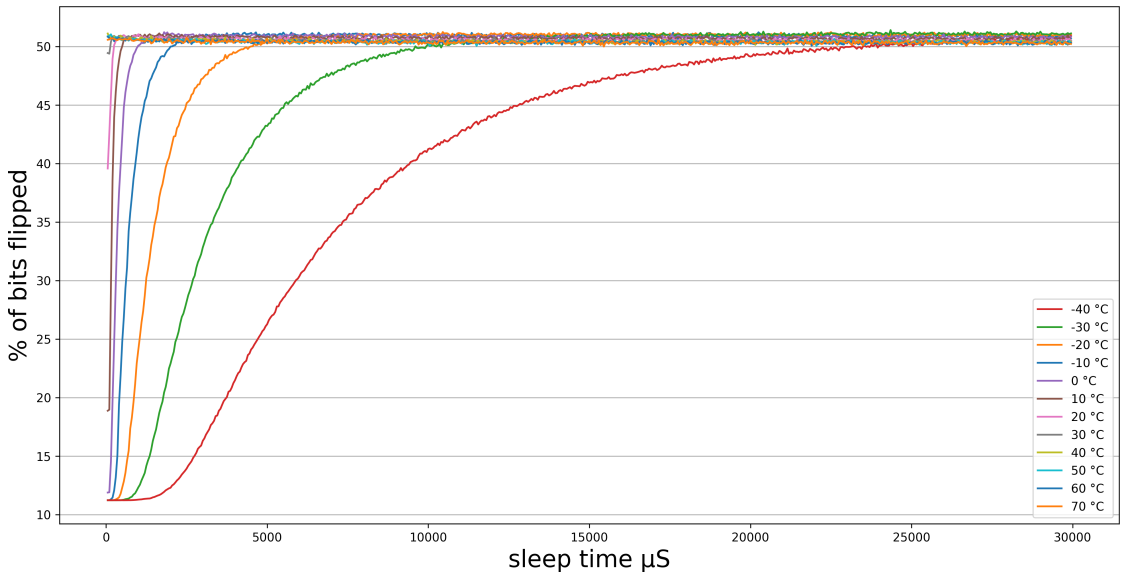


Figure 4.6 Percentage of flipped bits of MCU 6 across a range of temperatures (deep sleep method)

4.2.3 PUF evaluation parameters

The PUF responses needed for the calculation of the evaluation parameters were obtained by the deep sleep power-control method. The responses were chosen to be 4 KB bit strings of the internal SRAM 2 startup values. The turn off interval was set to 100 000 μS since the maximal *fade-out time* of memory was found to be 50 000 μS plus a conservative safety margin. The reference response was calculated using a bitwise average of 1000 measured responses.

Uniformity

Uniformity of all MCUs across the whole range of temperatures can be seen in Table 4.4. All of the results are close to the ideal value of 50 %, though a small bias towards the '1' bit is evident (all of the results are slightly above 50 %).

Reliability

Reliability values of all MCU in different temperatures are displayed in Table 4.5. The reference response was calculated at 20 °C.

MCU	-40°C	-30°C	-20°C	-10°C	0°C	10°C	20°C	30°C	50°C	70°C
1	51.46	51.33	51.14	51.01	50.90	50.75	50.70	50.67	50.59	50.54
2	51.55	51.43	51.25	51.06	50.94	50.82	50.67	50.63	50.56	50.53
3	51.09	50.97	50.84	50.72	50.60	50.47	50.20	50.18	50.19	50.22
4	50.66	50.62	50.49	50.31	50.22	50.19	50.17	50.15	50.08	50.00
5	50.96	50.83	50.72	50.58	50.49	50.38	50.38	50.33	50.27	50.22
6	51.19	51.06	50.90	50.82	50.71	50.61	50.46	50.37	50.30	50.28
7	51.01	51.02	50.95	50.85	50.73	50.68	50.54	50.49	50.40	50.37
8	50.95	50.86	50.73	50.61	50.47	50.37	50.23	50.16	50.01	49.94
9	51.54	51.36	51.24	51.14	51.03	50.91	50.61	50.53	50.39	50.40
10	51.37	51.18	51.03	50.89	50.78	50.68	50.50	50.48	50.36	50.37
11	51.63	51.54	51.40	51.28	51.13	51.06	50.86	50.86	50.81	50.81
12	51.56	51.40	51.23	51.06	50.93	50.77	50.57	50.42	50.27	50.24
13	50.81	50.71	50.60	50.55	50.50	50.48	50.42	50.34	50.22	50.15
14	50.26	50.17	50.11	50.11	50.03	50.03	50.02	50.05	50.14	50.22
15	51.55	51.45	51.36	51.32	51.26	51.16	51.08	50.98	50.79	50.64
16	52.43	52.31	52.24	52.18	52.07	51.89	51.81	51.66	51.26	50.92
mean	51.25	51.14	51.01	50.91	50.80	50.70	50.58	50.52	50.42	50.37

Table 4.4 Uniformity (%) across all MCUs and temperatures (deep sleep method)

The best reliability is achieved at 20 °C since the reference response was calculated at this temperature. Reliability starts to decrease in different temperatures. The decrease is proportional to the distance from the 20 °C of the reference response. However, the values are still quite close to the ideal 100 % and they only rarely fall under 90 % (only at -40 °C).

MCU	-40°C	-30°C	-20°C	-10°C	0°C	10°C	20°C	30°C	50°C	70°C
1	90.36	91.60	92.80	93.99	95.03	95.62	96.13	95.77	94.47	92.99
2	91.57	92.53	93.49	94.43	95.34	96.04	96.59	96.16	94.61	93.04
3	90.19	91.39	92.55	93.67	94.76	95.53	96.15	95.72	94.17	92.55
4	90.78	91.97	93.10	94.10	95.02	95.60	96.10	95.74	94.39	92.90
5	90.72	91.81	92.93	94.05	95.10	95.79	96.28	95.90	94.57	93.15
6	90.98	92.23	93.40	94.44	95.27	95.80	96.21	95.91	94.75	93.44
7	90.29	91.39	92.55	93.67	94.73	95.49	96.03	95.68	94.24	92.40
8	90.81	91.91	93.03	94.10	95.02	95.67	96.16	95.72	94.14	92.26
9	90.91	91.98	92.96	93.96	94.92	95.67	96.28	95.85	94.13	92.40
10	90.01	91.21	92.43	93.68	94.84	95.61	96.11	95.73	94.27	92.54
11	89.87	91.19	92.56	93.93	95.07	95.80	96.24	95.88	94.60	93.24
12	90.82	91.96	93.10	94.20	95.19	95.85	96.28	95.93	94.51	92.83
13	90.83	91.95	93.12	94.29	95.34	96.06	96.34	95.93	94.41	92.87
14	90.70	91.91	93.17	94.34	95.35	95.94	96.14	95.70	94.27	92.67
15	90.21	91.46	92.66	93.81	94.91	95.72	96.01	95.61	94.15	92.46
16	89.82	91.02	92.25	93.52	94.79	95.83	96.26	95.64	93.16	90.24
mean	90.55	91.72	92.88	94.01	95.04	95.75	96.21	95.80	94.30	92.62

Table 4.5 Reliability (%) across all MCUs and temperatures (deep sleep method)

Uniqueness

Uniqueness for all MCUs across the range of different temperatures is shown in Table 4.6. The results hover just under the ideal value of 50 % for all temperatures.

Temperature °C	Uniqueness %
-40	49.37
-30	49.41
-20	49.45
-10	49.49
0	49.51
10	49.56
20	49.61
30	49.63
40	49.66
50	49.68
60	49.70
70	49.73
mean	49.57

Table 4.6 Uniqueness (%) across all MCUs and temperatures (deep sleep method)

4.2.4 Summary

The presented experiments show very good results across the whole tested range of temperatures. The deep sleep functionality is available to all microcontrollers in the ESP32 family. This power-control method is therefore not restricted only to a particular subset of models. Additionally, most of the internal SRAM memory (around 400 KB) is available to be used in the PUF implementation.

On the other hand, the deep sleep power-control method is especially time-consuming (in comparison to the RTC SRAM method). This is mainly due to the need to go through the whole booting process. Obtaining the response can take up to several tenths of seconds, depending on the application size and speed of the connected flash memory. Moreover, most of the device state is inevitably lost during deep sleep. The application must save all data it does not want to lose to either the FAST or SLOW RTC SRAM (the only memory regions not turned off). Although, they together only provide 16 KB of space

4.3 PUF response data visualization

The above-described experiments and evaluation parameters only looked at the response data globally and reduced the entire response bit string to a single number. As will become clear, it can be useful to look at the data more locally. Therefore the following visualization is presented.

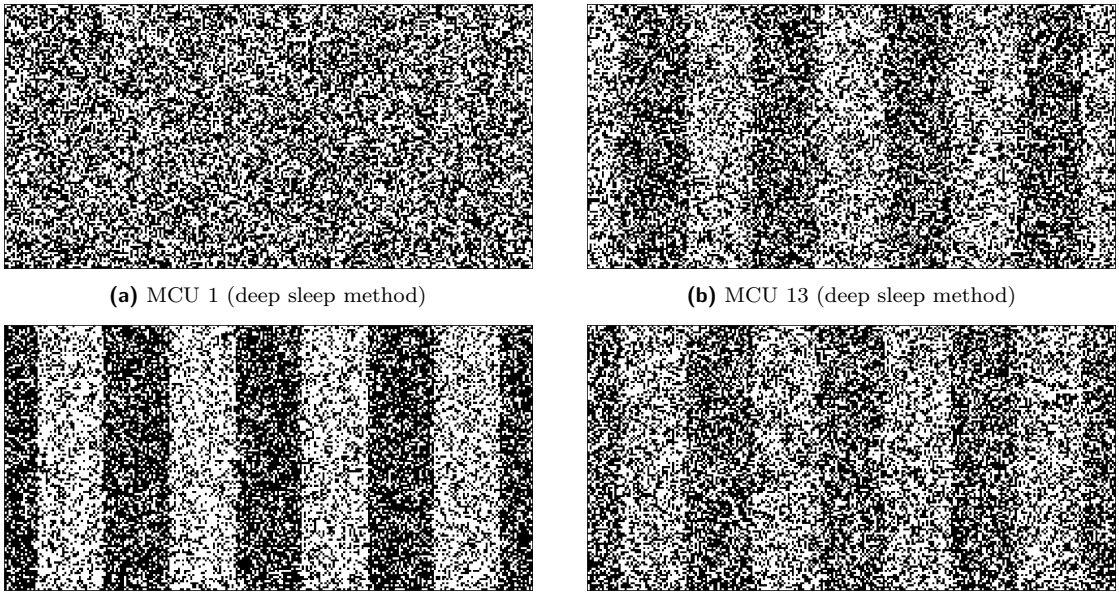
The whole 4 KB PUF response is arranged bit by bit to a 256×128 grid. Black bits represent the '0' state and white bits represent the '1' state. This visualization of 4 responses from different MCUs can be seen in Figure 4.7. The first two responses were obtained using the deep sleep method and the last two using the RTC SRAM method.

The response from MCU 1 (4.7a) looks as expected, the bits are arranged in a random-looking pattern[6]. However, the other responses show strange stripes. Every 32 bits the startup bias of memory cells towards a certain state flips. This results in this stripe-looking pattern. The

[6] This, of course, is not proof that the bits really are random

previous experiments did not reveal this behavior (mainly uniformity), as the overall number of '0' and '1' bits stays the same globally.

Both power-control methods produce the same result. Additionally, some of the MCUs do not exhibit this behavior at all (4.7a), some of them show only a slight 'stripe-like' bias (4.7b, 4.7d), and some are affected in a lot (4.7c).



(a) MCU 1 (deep sleep method)

(b) MCU 13 (deep sleep method)

(c) MCU 11 (RTC SRAM method)

(d) MCU 3 (RTC SRAM method)

Figure 4.7 Visualization of PUF responses from different MCUs

One edge case was also revealed. MCU 16 exhibited different behavior from all of the other MCUs. Its 'stripes' are 256 bits wide. In order to visualize this, the grid was reshaped into a 512×64 configuration which is shown in Figure 4.8.

However, only the RTC SRAM method shows this wide stripe behavior. This is probably not due to the different power-state control of memory, but rather because a different memory region is used for the response extraction.

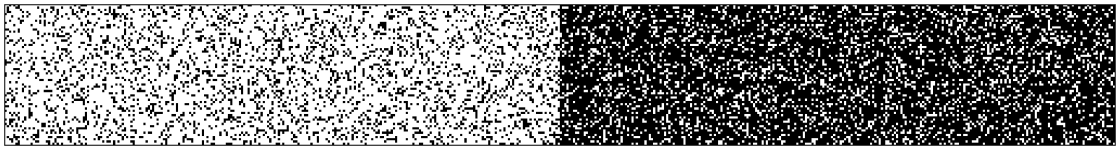


Figure 4.8 Visualization of PUF response from MCU 16 (RTC SRAM method).

This behavior is most likely caused by the internal design of the SRAM circuit and there is probably no way to prevent it. This explanation unfortunately cannot be confirmed as the ESP32 chip schematics are not available. It is, however, clear that the entropy of the resulting PUF response is reduced because of this behavior.

In order to improve the statistical properties of the PUF responses, a cryptographic hash function can be applied before using it. This is especially important if the PUF is used to generate a cryptographic key. [50], [51]

Chapter 5

Reliable PUF response reconstruction

Since SRAM PUFs are classified as weak (they do not have a large challenge-response pair set), their main use is for cryptographic key generation. The keys need to be extracted without error, otherwise the cryptographic algorithms that use the keys will not work. Therefore, methods of reliable response reconstruction are used to stabilize the PUF response.

In this chapter, stable bit preselection and ECC techniques are used on top of the designed memory power-control methods in order to stabilize the response of the ESP32 SRAM PUF. Then, a PUF design that combines the two power-control methods is presented. At the end, the enrollment process of key generation is discussed and the resulting SRAM PUF based key generation is tested on all MCUs at different operating temperatures. PUF design with only a single response will be implemented.

5.1 Stable bit preselection

Stable bit preselection consists of two steps. First, during the enrollment phase, the stable bits are selected and a stable bit mask is created. The mask is then saved to a non-volatile memory of the device. When a PUF response needs to be reconstructed, the mask is loaded and the unstable bits are discarded.

Direct bit preselection was used to obtain the stable bits. Other methods that try to preselect stable bits exist, such as the indirect preselection method described in Section 2.2. However, the discussed stability test which measures the stabilization time of each memory cell, cannot be used for our PUF implementation. The deep sleep method of memory power-control does not enable turn off intervals of less than 1000 μS. Thus, it is not possible to carry out the test accurately.

Stable bit mask creation

The stable bit mask is created by measuring a PUF response multiple times. An average startup value $\bar{x}$ of each bit is then calculated from the responses. Then, a threshold value, called the error rate E_r, needs to be selected. Bits with average value $\bar{x} \leq E_r$ (stable '0') or $\bar{x} \geq (1 - E_r)$ (stable '1') are declared as stable.

Two techniques were used to generate the stable bit mask. First, 1000 measurements of the PUF response were taken in 20 °C and the mask was created using the above-described method.

The second technique was inspired by [52]. Twelve different intermediate masks were created, each from 1000 measurements of the PUF response. The measurements for each mask were captured in different operating conditions.[1] Then, the final mask was calculated as a bitwise logical AND from the twelve intermediate masks—a bit is declared stable if it is stable at all twelve operating temperatures.

Results

Table 5.1 shows the percentage of stable bits found in a 4 KB sample PUF response for all MCUs. Three different error rates were chosen, 10 %, 1 % and 0.1 %. As expected, lower error rate reduces the number of stable bits. Moreover, the temperature range mask (created by the second technique) further reduced the percentage of stable bits by about 17 % compared to the 20 °C mask. Only the results for the deep sleep method of power-control are presented, as the temperature range mask cannot be used for the RTC SRAM method (because of the memory freezing effect). The error rate of 0.1 % was chosen for the SRAM PUF implementation as it provides the best reliability with only a minor reduction of available bits.

	20 °C mask			**Temperature range mask**		
MCU	**10 %**	**1.0 %**	**0.1 %**	**10 %**	**1.0 %**	**0.1 %**
1	87.37	77.32	71.24	72.39	62.87	56.43
2	87.56	77.66	71.58	72.79	63.05	56.67
3	87.94	78.60	72.82	73.54	64.29	58.24
4	87.55	77.50	71.34	72.31	62.90	56.76
5	87.75	77.91	71.82	74.41	64.91	58.47
6	87.89	78.09	72.11	73.18	63.72	57.80
7	88.11	78.97	72.93	73.93	64.56	58.34
8	87.58	77.89	72.04	73.93	64.59	58.52
9	88.03	78.49	72.38	74.12	64.55	58.25
10	87.70	77.88	71.83	73.43	63.84	57.54
11	89.07	80.03	74.57	75.36	66.24	60.24
12	87.56	77.60	71.51	72.64	62.93	56.76
13	88.22	79.13	73.32	73.89	65.01	59.09
14	87.50	77.67	71.80	73.04	63.68	57.60
15	87.30	77.18	71.51	72.42	62.95	56.71
16	88.00	78.59	72.84	70.15	61.36	55.54
mean	87.82	78.16	72.23	73.22	63.84	57.68

Table 5.1 Percentage of stable bits across all MCUs. Three different error rates (10 %, 1 % and 0.1 %) and two stable bit selection methods were used.

Table 5.2 shows reliability of PUF responses at different operating temperatures and for all MCUs. The reference response was obtained at 20 °C and only the stable bits of the responses were selected using the 20 °C mask. The table shows results using the deep sleep power-control method, but the RTC SRAM method shows nearly identical results above -10 °C.

The best reliability is observed at 20 °C, since this is the temperature the reference and stable bit mask were calculated at. With increasing distance from 20 °C, the reliability reduces. However, reliability was increased significantly thanks to the stable bit preselection. Reliability results without the preselection can be found in Table 4.5.

Reliability table using the temperature range mask is not presented, as the reliability values for all temperatures and MCUs are above 99.99 %. This technique thus increases reliability significantly. Unfortunately, the creation of the temperature mask requires a controlled temperature environment and takes much longer, making the enrollment phase more difficult and costly. On

[1] The temperatures of -40, -30, -20, -10, 0, 10, 20, 30, 40, 50, 60 and 70 °C were used for the twelve masks.

MCU	-40°C	-30°C	-20°C	-10°C	0°C	10°C	20°C	30°C	50°C	70°C
1	98.27	98.96	99.45	99.75	99.88	99.92	99.99	99.97	99.83	99.42
2	98.59	99.09	99.45	99.71	99.87	99.92	99.99	99.95	99.74	99.19
3	98.17	98.84	99.33	99.65	99.81	99.89	99.99	99.97	99.81	99.30
4	98.51	99.11	99.51	99.76	99.88	99.92	99.99	99.97	99.83	99.44
5	98.24	98.87	99.37	99.71	99.87	99.93	99.99	99.98	99.84	99.44
6	98.50	99.13	99.54	99.76	99.88	99.91	99.99	99.97	99.87	99.58
7	98.21	98.86	99.37	99.68	99.85	99.92	99.99	99.97	99.77	99.15
8	98.31	98.92	99.36	99.65	99.81	99.89	99.99	99.95	99.74	99.08
9	98.51	99.06	99.44	99.70	99.84	99.90	99.99	99.96	99.74	99.09
10	98.14	98.82	99.36	99.71	99.88	99.93	99.99	99.97	99.79	99.22
11	97.99	98.78	99.39	99.75	99.91	99.94	99.99	99.98	99.83	99.45
12	98.32	98.98	99.46	99.74	99.88	99.92	99.99	99.98	99.79	99.23
13	98.29	98.95	99.46	99.79	99.94	99.97	99.99	99.97	99.78	99.27
14	98.33	99.03	99.54	99.83	99.94	99.97	99.99	99.95	99.79	99.30
15	98.22	98.95	99.45	99.76	99.92	99.97	99.99	99.97	99.80	99.30
16	97.41	98.20	98.88	99.42	99.80	99.96	99.99	99.96	99.33	97.60
mean	98.25	98.91	99.40	99.71	99.87	99.93	99.99	99.97	99.77	99.19

Table 5.2 Reliability (%) across all MCUs and temperatures (deep sleep method). Preselected stable bits with error rate of 0.1 % were used

the other hand, the 20 °C mask can be calculated easily (the exact value of temperature does not matter and room temperature suffices).

5.2 Error correction code

Stable bit preselection managed to increase reliability significantly. However, for successful key reconstruction the reliability must be 100 % as only a single bit flip renders the key unusable. For this reason, ECCs are used as an additional method to reduce the noise of the PUF responses.

One method of applying an ECC to the PUF responses is called the code-offset construction. In the enrollment phase, a reference response needs to be chosen. A random code word of the used code is then subtracted from the response and the results are stored in a non-volatile memory. This data is called the helper data. During the key reconstruction phase, a new, potentially noisy PUF response is generated. Helper data are retrieved from memory and added to the response, creating the code word of the ECC. This (also potentially noisy) code word is then decoded by the ECC, correcting the errors. [50], [51]

The repetition code of length 8 was chosen for the SRAM PUF implementation. It is very simple to implement and the encoding and decoding operations are extremely fast, because the code words are byte-sized. It is capable of correcting up to 3 errors and can detect up to 4 errors per byte.

The above-described code-offset construction is used for extracting the key. Addition and subtraction are both realized as the logical XOR operation. Each eight bits of the reference response are XORed with a code word chosen by the first bit of the eight, generating the helper data.[2] During reconstruction, the helper data is again XORed with the new noisy PUF response and the resulting code words are decoded using a majority bit. This process is illustrated in Figure 5.1, by using a shorter repetition code of length 5.

Since the ECC can correct only a limited number of errors, the PUF response can be too noisy for the key to be properly reconstructed. The probability of this happening can be modeled using the binomial distribution. Assuming that all response bits are independent and have the

[2] This ECC only has two code words, 00000000 and 11111111.

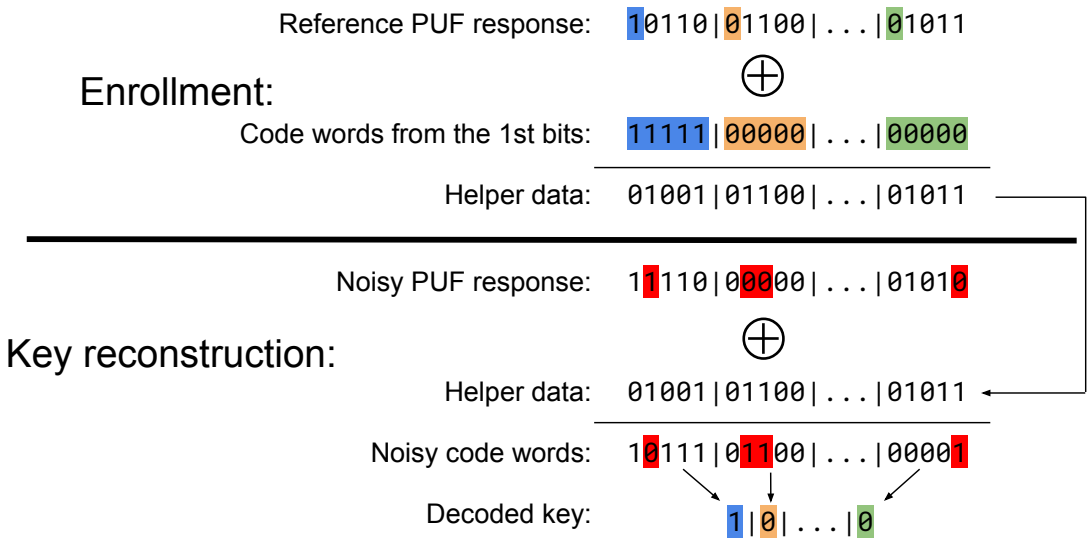


Figure 5.1 An example of enrollment and key reconstruction phases for a repetition code of length 5. Red bits are the error bits (flipped compared to the reference response), other colors represent individual bits of the key. [6]

same probability of error[3], the probability of successfully decoding a single code word can be calculated using the binomial cumulative distribution function: [53], [26]

$$F(k,n,p) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p^{n-i}) \tag{5.1}$$

N is the size of the code word, k is the maximum number of corrected errors and p is the probability of a single bit error. The Equation 5.1 only calculates the probability of successfully decoding a single code word which gives us only one bit of the key (because of the repetition code). To obtain the whole key, each bit is reconstructed independently and thus the probability of successful reconstruction P_{succ} can be calculated using the Equation 5.2, where B is the number of bits of the key (or equivalently the number of bytes of the reference response).

$$P_{\text{succ}} = F(k,n,p)^B \tag{5.2}$$

For our case of the repetition code of length 8, maximum of three errors can be corrected in 8 bits. Therefore $n = 8$, $k = 3$ and p can be estimated by the intra-Hamming distance or $1-$reliability. [53] B was chosen as $4096 \times 0.7223 \approx 2958$: a 4 KB response and 72.23 % of stable bits (a mean value across all MCUs using the 20 °C mask).

[26] states that key reconstruction should achieve P_{succ} of at least $1 - 10^{-6}$ or less than one error in a million. In order to meet this requirement, the value of reliability must be greater than 99.85 % according to the equations above. Using this knowledge, it can be concluded that a SRAM PUF implementation using the 20°C stable bit mask and deep sleep power-control method is suitable only in temperatures between approximately 0 and 40 °C (see Table 5.2). A stronger ECC needs to be used to extend the temperature range. As the temperature range mask achieves reliability greater than 99.99 % in all temperatures, it is suitable to be used in all tested conditions with this ECC.

[3]Error is understood as a bit flip compared to the reference PUF response.

This model further calculates that P_{succ} is 98.181 % (or about 1 error in 55) for -40 °C and 99.913 % (or about 1 error in 1150) for 70 °C.[4] However, the real-world testing showed considerably lower number of errors in key reconstructions than was predicted by this model. Results from this testing can be found in Chapter 5.4

The repetition code of length eight reduces the size of the PUF response by a factor of eight. The average length of the reconstructed response thus is 2958 bits for the 20 °C stable bit mask (minimum of 2917 bits was observed for MCU 1) and 2362 bits for the temperature range mask (minumum of 2274 bits was observed for MCU 16). The size of the response can be doubled if 8 KB of SRAM data is used instead of 4 KB. The 8 KB is however the limit for the RTC SRAM as this is the size of the memory segment used in this method.

5.3 Combining power-control methods

Two different SRAM power-control methods were introduced in Chapter 4. The RTC SRAM method enables fast response extraction, but works reliably only in temperatures above 0°C. On the other hand, the deep sleep method is time-consuming but reliable. In this section, a design that leverages the advantages and reduces the disadvantages of the two methods is presented by combining them.

The idea is, that the RTC SRAM method will be used for temperatures it works reliably at and the deep sleep method will play a role of a fallback mechanism. For this to work, both of the methods need to reconstruct the same PUF response. The startup memory values are inherently different, as they are obtained from separate regions of memory. So to reconstruct the same PUF response every time, the generation of the ECC helper data is altered for the RTC SRAM method.

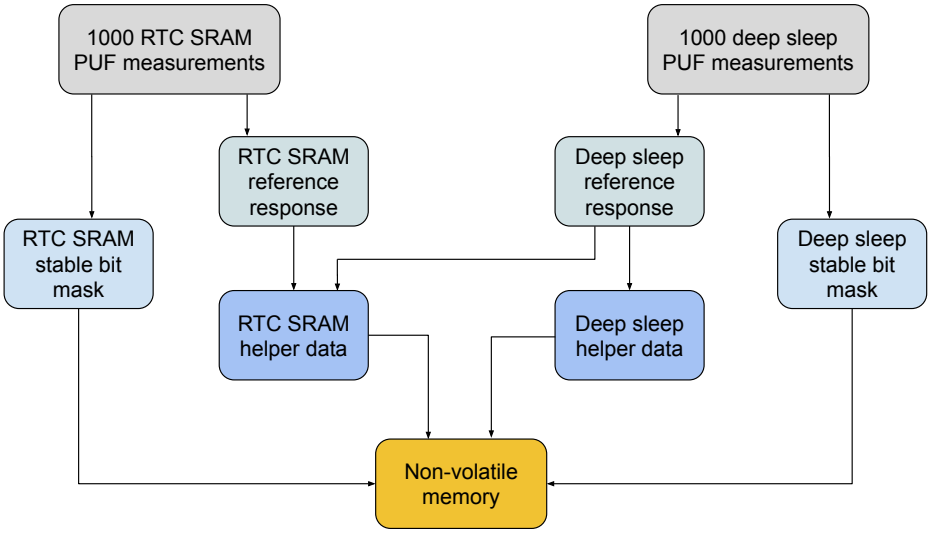


Figure 5.2 Enrollment diagram using the combination of both power-control methods to obtain the same PUF response

Enrollment phase

Normally, each bit of the PUF response is reconstructed from a single code word of the repetition code. This code word is determined by the first bit of the corresponding byte of the reference

[4] Again, using the 20 °C stable bit mask and deep sleep power-control method.

response. The code word is then XORed with the given byte of the reference, creating the helper data (as illustrated in Figure 5.1). So to obtain the same response after ECC decoding, the code words for each byte of the RTC SRAM are chosen not by its reference response, but by the reference of the deep sleep method. The resulting enrollment process diagram is shown in Figure 5.2.

Power-control method selection

In order to choose which method to use, a detection mechanism of the freezing effect of the RTC SRAM needs to be implemented. Two indicators were chosen. The Hamming weight of the startup memory values (the memory is initialized to '0' bits before power off and about 50 % of bits are expected to flip after power on) and the number of errors detected by the ECC. Low Hamming weight or high error count indicate that the RTC SRAM memory froze because of low temperature and the deep sleep method needs to be used.

The mathematical model from Section 5.2 tells us, that reliability needs to be at least 99.85 %. This is equivalent to HD-intra < 0.15 % (or about 35 bit errors) as reliability = 100% - HD-intra. Since HD-intra represents the percentage of bits that flip compared to the reference response, it can be estimated by the percentage of errors detected by the ECC.

The Hamming weight threshold was chosen as 48.5 %, which is lower than the uniformity values from Table 4.1 for non-frozen memory.

Experimental verification of the threshold values

In order to confirm that the selected threshold values are viable and really detect the effect of memory freezing, the following experiment was conducted. Six MCUs were continuously reconstructing their PUF responses using only the RTC SRAM method. Three data points were recorded for each reconstruction—the ECC error detection count, the Hamming weight of the startup data and a correct reconstruction flag. The operating temperature was slowly increased from -40 to 70 °C. The resulting scatter plot can be seen in Figure 5.3.

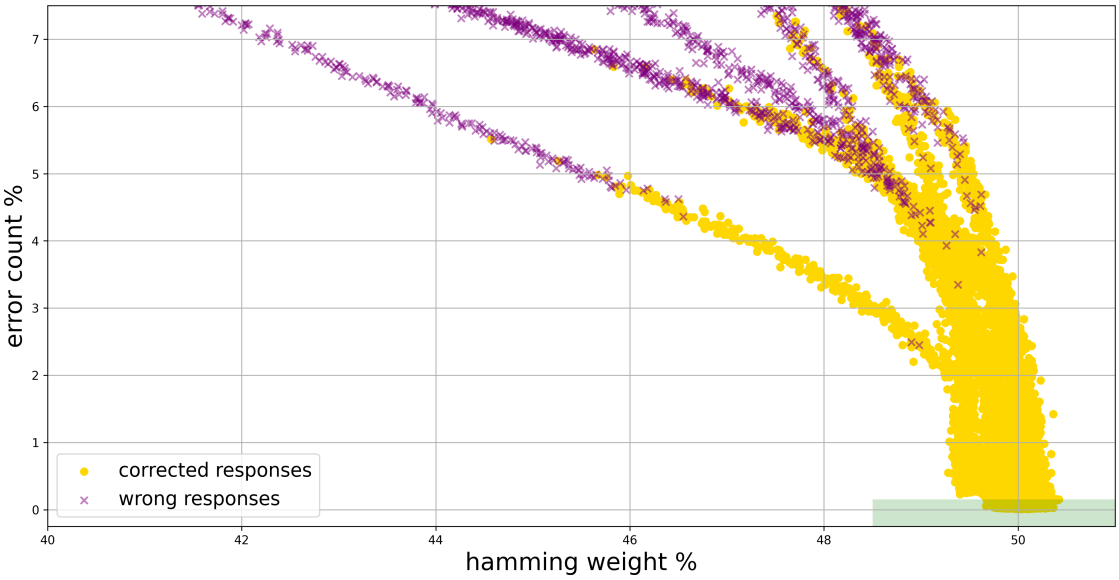


Figure 5.3 A scatter plot showing the effect of Hamming weight and error count on correct response reconstruction. The green region represents data points where the RTC SRAM response is accepted as a valid response. Data obtained from MCUs 1–6.

The scatter plot is zoomed-in on the part where, as the temperature increases, the responses are starting to be reconstructed properly. This happens at slightly different temperatures for each MCU, but generally at about -5°C (the exact temperature does not matter since the threshold values do not depend on it). As expected, the error count and Hamming weight are highly negatively correlated. As the temperature increases, the Hamming weight increases and error count decreases. Once the error count is low enough (and the Hamming weight high enough), the responses can be correctly reconstructed. The lowest error rate, where the response was not reconstructed properly, is 2.45 % (or about 580 bit errors). This proves that there is an extremely large safety margin.

Resulting design

First, the enrollment phase is executed as is illustrated in Figure 5.2. The resulting procedure of obtaining the PUF response is then the following:

1. Try to reconstruct the PUF response using the RTC SRAM power-control method. Calculate the Hamming weight and the ECC detected error percentage in the process.

2. If the Hamming weight is above the threshold of 48.5 % and the error percentage is lower than the threshold of 0.15 %, use the reconstructed PUF response as a result and end.

3. Reconstruct the PUF response using the deep sleep method (since the RTC SRAM method could have produced incorrect response).

5.4 Reliability testing

In this section, the results of testing of SRAM PUF with combined power-control and reliable response reconstruction are presented. The response reconstruction process was executed 1000 times at each temperature for all MCUs. Table 5.3 contains the percentages of successful response reconstructions using the 20 °C stable bit mask.

MCU	-40°C	-30°C	-20°C	-10°C	0°C	10°C	20°C	30°C	50°C	70°C
1	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
3	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
4	98.5	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
5	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
6	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
7	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
8	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
10	98.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
11	98.8	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
12	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
13	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
14	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
15	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
16	99.2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	94.7
mean	99.63	99.99	100.0	100.0	100.0	100.0	100.0	100.0	100.0	99.67

Table 5.3 Success rate (%) of reliable response extraction across all MCUs and temperatures. The 20 °C stable bit mask was used for stable bit preselection.

The outlier in this experiment is the MCU 16. It is the only one with incorrect reconstructions at 70 °C (and the success rate is quite low too). As this is the same MCU as the one from Figure 4.8, lower success rate of this chip may be linked to its weird startup memory behavior.

Results using the temperature range mask are not presented, as all MCUs reached 100 % success rate at all temperatures. This matches expectations of the stable bits reliability results (all were above 99.99 %).

Furthermore, the RTC SRAM method stops working at about -5 °C and 100 % success rate was observed around this temperature. This proves that the fallback mechanism of the combined power-control PUF works reliably.

Chapter 6

ESP32 SRAM PUF library

The presented SRAM PUF design with combined power-control and stable PUF response reconstruction using repetition ECC and stable bit preselection was implemented in a library called esp32_puflib.

The esp32_puflib library is implemented as an ESP-IDF component. Components are modular pieces od standalone code. They are compiled as a static library and then linked into the main project by the toolchain. The ESP-IDF component manager makes it easy to add existing components into the main project. [44]

6.1 Enrollment

In order to be able to reconstruct the PUF responses properly, the device needs to be enrolled first. This means creating the stable bit mask, calculating the ECC helper data and saving them to non-volatile memory. Two different enrollment procedures were implemented, in-device enrollment and external enrollment.

In-device enrollment

In the in-device enrollment, the ESP32 microcontroller does all of the needed calculations by itself. It is performed by the `provision_puf` function. This method of enrollment is very straightforward to use (just a function call).

Unfortunately, the stable bit mask can be calculated only in one operating temperature (which results in erroneous response reconstructions at extreme temperatures). Furthermore, the in-device enrollment process is lengthy. Stable bit mask created from 1000 PUF measurements takes about 40 minutes to calculate—each measurement executes the deep sleep restart and a write the non-volatile flash memory to save the results. Thus this method also increases wear of the flash memory.

External enrollment

In the external enrollment, the ESP32 sends the PUF measurements to a server through UART. The measurements can be repeated in different operating temperatures to increase stability. The server then calculates stable bit mask and the ECC helper data from the received measurements. The results are then exported as a non-volatile storage (NVS) partition[1] and uploaded to the device.

[1] NVS is a ESP-IDF library for saving data to flash memory.

This method is quicker (with 1000 PUF measurements the process takes about 5 minutes) and enables measurements in different operating conditions, increasing stability significantly. On the other hand, it requires a more elaborate setup (serial connection with the server that runs the enrollment scripts).

6.2 Example of usage

The minimal working example code of the esp32_puflib library is shown in Code listing 6.1. The code initializes the library and reconstructs the PUF response. If the RTC SRAM method did not succeed in the reconstruction, the deep sleep method is used. In this case, the application needs to save its state and is then restarted (line 17). The PUF response is then printed to `stdout` and erased from memory.

After reconstructing the response (either by the RTC SRAM method immediately or after restart by the deep sleep method), the library sets global variables with the PUF response accordingly—`PUF_STATE` flag to `RESPONSE_READY`, `PUF_RESPONSE` to the response data and `PUF_RESPONSE_LEN` to response length in bytes.

```
1  #include <stdio.h>
2  #include <esp_sleep.h>
3  #include <puflib.h>
4
5  void app_main(void) {
6      // needs to be called first in app_main
7      puflib_init();
8      // provisioning needs to be done only once at the beginning
9      provision_puf();
10
11     // condition will be true, if a PUF response is ready
12     // (useful after a restart)
13     if(PUF_STATE != RESPONSE_READY) {
14         bool puf_ok = get_puf_response();
15         if(!puf_ok) {
16             // the device resets now, SAVE APPLICATION STATE
17             get_puf_response_reset();
18         }
19     }
20
21     // PUF_RESPONSE_LEN is a PUF response length in bytes
22     for (size_t i = 0; i < PUF_RESPONSE_LEN; ++i) {
23         // PUF_RESPONSE is a buffer with the PUF response
24         printf("%02X ", PUF_RESPONSE[i]);
25     }
26
27     // erases the PUF response from memory
28     clean_puf_response();
29 }
30
31 void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
32     esp_default_wake_deep_sleep();
33     // needs to be called somewhere in wake up stub
34     puflib_wake_up_stub();
35 }
```

Code listing 6.1 Minimal working example of the ESP32_puflib library

Conclusion

In this thesis, the possibility of implementing a SRAM PUF on the ESP32 microcontroller was analyzed and a proof of concept solution was implemented.

The first chapter provides a description of the concept of a PUF. Then, PUF properties, applications, classification and evaluation parameters were described. At the end, three PUF implementations were presented.

A thorough explanation of SRAM PUFs was given in the second chapter. A discussion on which properties the SRAM PUF possesses was provided. Then, stable bit preselection methods and the effect of silicon aging were presented. At the end, existing research on SRAM PUFs on the ESP32 platform was summarized.

In the third chapter, an overview of the ESP32 IoT platform was given. Hardware and software components of the ESP32 microcontroller with a focus on features specific to the SRAM PUF implementation were introduced.

Two SRAM power state control methods on the ESP32 microcontroller were designed in the fourth chapter. Their resulting PUF responses were analyzed based on operating temperature and power-off time of the memory. This analysis detected an interesting effect of memory 'freezing' for the RTC SRAM method in low operating temperatures. Then, evaluation parameters were used to test the PUF responses and the results matched the expectations from the previous analysis. The advantages and disadvantages of each method and their suitability for use were also discussed. At the end, a visualization of the PUF responses was presented, revealing an interesting pattern in the memory startup values.

Chapter five implemented reliable PUF response reconstruction using two methods of stable bit preselection and a simple repetition ECC. Next, a PUF design that combines the two power-control methods was presented. At the end, the proposed SRAM PUF design was tested on 16 MCUs in different operating temperatures.

The final chapter introduced esp32_puflib—a library that implements the proposed SRAM PUF design.

To summarize, all of the goals outlined were accomplished, resulting in interesting findings and a working SRAM PUF implementation on the ESP32 microcontroller.

Future work

At the end, ideas for possible future research are presented:

- test the PUF responses using more evaluation parameters (such as randomness or bit-aliasing)
- use more advanced ECCs to achieve greater reliability without the need of sophisticated stable bit preselection
- evaluate the behaviour of the PUF depending on other operating conditions (for example supply voltage)
- test the SRAM PUF implementation on the new RISC-V based ESP32 microcontroller models
- apply the reconstructed key to secure the ESP32 wireless communication

Bibliography

1. MAES, Roel. Physically Unclonable Functions: Properties. In: *Physically Unclonable Functions: Constructions, Properties and Applications* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013 [visited on 2022-03-24]. ISBN 978-3-642-41395-7. Available from DOI: 10.1007/978-3-642-41395-7_3.
2. RAVIKANTH, Pappu Srinivasa. *Physical One-Way Functions* [online]. 2001 [visited on 2022-03-23]. Available from: http://alumni.media.mit.edu/~pappu/pdfs/Pappu-PhD-POWF-2001.pdf. PhD thesis. Massachusetts Institute of Technology.
3. GASSEND, Blaise; CLARKE, Dwaine; DIJK, Marten van; DEVADAS, Srinivas. Silicon Physical Random Functions. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security* [online]. Washington, DC, USA: Association for Computing Machinery, 2002 [visited on 2022-03-23]. CCS '02. ISBN 1581136129. Available from DOI: 10.1145/586110.586132.
4. KODÝTEK, Filip. *PUF založený na kruhových oscilátorech pro FPGA* [online]. 2020 [visited on 2022-03-23]. Available from: https://dspace.cvut.cz/handle/10467/94208. PhD thesis. Faculty of Information Technology, Czech Technical University in Prague.
5. MAES, Roel. *Physically Unclonable Functions: Constructions, Properties and Applications (Fysisch onkloonbare functies: constructies, eigenschappen en toepassingen)* [online]. 2012 [visited on 2022-03-24]. Available from: https://lirias.kuleuven.be/retrieve/192818. PhD thesis. Katholieke Universiteit Leuven – Faculty of Engineering.
6. KODÝTEK, Filip. *HW Bezpečnost - Fyzicky neklonovatelné funkce* [online]. Faculty of Information Technology, Czech Technical University in Prague, 2017 [visited on 2022-04-04]. Available from: https://courses.fit.cvut.cz/BI-HWB/@B172/media/lectures/07/prednaska-puf-cz.pdf.
7. KÖMMERLING, Oliver; KUHN, Markus G. Design Principles for Tamper-Resistant Smartcard Processors. In: *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology* [online]. Chicago, Illinois: USENIX Association, 1999 [visited on 2022-03-27]. WOST'99. Available from: https://www.cl.cam.ac.uk/~mgk25/sc99-tamper.pdf.
8. JOSHI, Shital; MOHANTY, Saraju; KOUGIANOS, Elias. Everything You Wanted to Know about PUFs. *IEEE Potentials* [online]. 2017, vol. 36 [visited on 2022-03-27]. Available from DOI: 10.1109/MPOT.2015.2490261.
9. MCGRATH, Thomas; BAGCI, Ibrahim Ethem; WANG, Zhiming M.; ROEDIG, Utz. A PUF taxonomy. *Applied Physics Reviews* [online]. 2019 [visited on 2022-03-27]. Available from DOI: 10.1063/1.5079407.

10. PHALAK, Koustubh; SAKI, Abdullah Ash-; ALAM, Mahabubul; TOPALOGLU, Rasit Onur; GHOSH, Swaroop. Quantum PUF for Security and Trust in Quantum Computing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* [online]. 2021, vol. 11, no. 2, pp. 333–342 [visited on 2022-03-27]. Available from DOI: 10.1109/JETCAS.2021.3077024.

11. MAES, Roel; VERBAUWHEDE, Ingrid. Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. In: [online]. 2010 [visited on 2022-04-07]. ISBN 978-3-642-14451-6. Available from DOI: 10.1007/978-3-642-14452-3_1.

12. SKORIC, Boris; MAUBACH, Stefan; KEVENAAR, Tam Tom; TUYLS, Pim. Information-theoretic analysis of capacitive physical unclonable functions. *Journal of Applied Physics* [online]. 2006, vol. 100 [visited on 2022-03-27]. Available from DOI: 10.1063/1.2209532.

13. GUAJARDO, Jorge; KUMAR, Sandeep S.; SCHRIJEN, Geert-Jan; TUYLS, Pim. FPGA Intrinsic PUFs and Their Use for IP Protection. In: PAILLIER, Pascal; VERBAUWHEDE, Ingrid (eds.). *Cryptographic Hardware and Embedded Systems - CHES 2007* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 65–68 [visited on 2022-03-28]. ISBN 978-3-540-74735-2. Available from DOI: 10.1007/978-3-540-74735-2_5.

14. MISPAN, Mohd Syafiq. *Towards reliable and secure physical unclonable functions* [online]. University of Southampton, 2018 [visited on 2022-03-30]. Available from: https://eprints.soton.ac.uk/424534/. PhD thesis. University of Southampton.

15. LEEST, Vincent van der; SCHRIJEN, Geert-Jan; HANDSCHUH, Helena; TUYLS, Pim. Hardware Intrinsic Security from D Flip-Flops. In: *Proceedings of the Fifth ACM Workshop on Scalable Trusted Computing* [online]. Chicago, Illinois, USA: Association for Computing Machinery, 2010 [visited on 2022-03-30]. STC '10. ISBN 9781450300957. Available from DOI: 10.1145/1867635.1867644.

16. MAITI, Abhranil; GUNREDDY, Vikash; SCHAUMONT, Patrick. A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. *IACR Cryptol. ePrint Arch.* [Online]. 2011 [visited on 2022-03-30]. Available from: https://eprint.iacr.org/2011/657.pdf.

17. MAITI, Abhranil; CASARONA, Jeff; MCHALE, Luke; SCHAUMONT, Patrick. A large scale characterization of RO-PUF. In: *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* [online]. 2010, pp. 94–99 [visited on 2022-04-04]. Available from DOI: 10.1109/HST.2010.5513108.

18. BASSHAM, Lawrence; RUKHIN, Andrew; SOTO, Juan; NECHVATAL, James; SMID, Miles; LEIGH, Stefan; LEVENSON, M; VANGEL, M; HECKERT, Nathanael; BANKS, D. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* [online]. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2010 [visited on 2022-04-04]. Available from: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762.

19. YUN, Gao; KONTOYIANNIS, Ioannis; ELIE, Bienenstock. Estimating the Entropy of Binary Time Series: Methodology, Some Theory and a Simulation Study. *Entropy: International and Interdisciplinary Journal of Entropy and Information Studies* [online]. 2008, vol. 10 [visited on 2022-04-04]. Available from DOI: 10.3390/entropy-e10020071.

20. HORI, Yohei; KANG, Hyunho; KATASHITA, Toshihiro; SATOH, Akashi. Pseudo-LFSR PUF: A Compact, Efficient and Reliable Physical Unclonable Function. In: *2011 International Conference on Reconfigurable Computing and FPGAs* [online]. Research Center for Information Security, National Institute of Advanced Industrial Science and Technology, 2011 [visited on 2022-04-06]. Available from DOI: 10.1109/ReConFig.2011.72.

21. DEVADAS, Srinivas; SUH, Edward; PARAL, Sid; SOWELL, Richard; ZIOLA, Tom; KHANDELWAL, Vivek. Design and Implementation of PUF-Based "Unclonable" RFID ICs for Anti-Counterfeiting and Security Applications. In: *2008 IEEE International Conference on RFID* [online]. 2008 [visited on 2022-04-06]. Available from DOI: 10.1109/RFID.2008.4519377.

22. BARBARESCHI, Mario; DE BENEDICTIS, Alessandra; MAZZOCCA, Nicola. A PUF-based hardware mutual authentication protocol. *Journal of Parallel and Distributed Computing* [online]. 2018, vol. 119 [visited on 2022-04-06]. ISSN 0743-7315. Available from DOI: https://doi.org/10.1016/j.jpdc.2018.04.007.

23. ROSTAMI, Masoud; MAJZOOBI, Mehrdad; KOUSHANFAR, Farinaz; WALLACH, Dan S.; DEVADAS, Srinivas. Robust and Reverse-Engineering Resilient PUF Authentication and Key-Exchange by Substring Matching. *IEEE Transactions on Emerging Topics in Computing* [online]. 2014, vol. 2, no. 1 [visited on 2022-04-06]. Available from DOI: 10.1109/TETC.2014.2300635.

24. MAES, Roel; VAN HERREWEGE, Anthony; VERBAUWHEDE, Ingrid. PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator. In: PROUFF, Emmanuel; SCHAUMONT, Patrick (eds.). *Cryptographic Hardware and Embedded Systems – CHES 2012* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012 [visited on 2022-04-07]. ISBN 978-3-642-33027-8. Available from DOI: 10.1007/978-3-642-33027-8_18.

25. GAO, Yansong; MA, Hua; LI, Geifei; ZEITOUNI, Shaza; AL-SARAWI, Said; ABBOTT, Derek; SADEGHI, Ahmad-Reza; RANASINGHE, Damith. Exploiting PUF Models for Error Free Response Generation [online]. 2017 [visited on 2022-04-06]. Available from DOI: 10.48550/arXiv.1701.08241.

26. BÖSCH, Christoph; GUAJARDO, Jorge; SADEGHI, Ahmad-Reza; SHOKROLLAHI, Jamshid; TUYLS, Pim. Efficient helper data key extractor on FPGAs. In: [online]. 2008, vol. 5154 [visited on 2022-04-07]. ISBN 978-3-540-85052-6. Available from DOI: 10.1007/978-3-540-85053-3_12.

27. HERDER, Charles; YU, Meng-Day; KOUSHANFAR, Farinaz; DEVADAS, Srinivas. Physical Unclonable Functions and Applications: A Tutorial. *Proceedings of the IEEE* [online]. 2014, vol. 102, no. 8 [visited on 2022-04-06]. Available from DOI: 10.1109/JPROC.2014.2320516.

28. PAPPU, Ravikanth; RECHT, Ben; TAYLOR, Jason; GERSHENFELD, Neil. Physical One-Way Functions. *Science* [online]. 2002, vol. 297, no. 5589 [visited on 2022-04-07]. Available from DOI: 10.1126/science.1074376.

29. RÜHRMAIR, Ulrich; HILGERS, Christian; URBAN, Sebastian; WEIERSHÄUSER, Agnes; DINTER, Elias; FORSTER, Brigitte; JIRAUSCHEK, Christian. *Optical PUFs Reloaded* [Cryptology ePrint Archive, Report 2013/215]. 2013 [visited on 2022-04-07]. Available from: https://eprint.iacr.org/2013/215.pdf.

30. LEE, J.W.; LIM, Daihyun; GASSEND, B.; SUH, G.E.; DIJK, M. van; DEVADAS, S. 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)* [online]. 2004 [visited on 2022-04-07]. Available from DOI: 10.1109/VLSIC.2004.1346548.

31. LIM, Daihyun; LEE, J.W.; GASSEND, B.; SUH, G.E.; DIJK, M. van; DEVADAS, S. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* [online]. 2005, vol. 13, no. 10 [visited on 2022-04-07]. Available from DOI: 10.1109/TVLSI.2005.859470.

32. TUYLS, Pim; SCHRIJEN, Geert; SKORIC, Boris; GELOVEN, Jan; VERHAEGH, Nynke; WOLTERS, Rob. Read-Proof Hardware from Protective Coatings. In: [online]. 2006 [visited on 2022-04-07]. ISBN 978-3-540-46559-1. Available from DOI: 10.1007/11894063_29.

33. VOINIGESCU, Sorin. *High-Frequency Integrated Circuits* [online]. Ed. by CRIPPS, Steve C. Cambridge University Press, 2013 [visited on 2022-04-08]. ISBN 9780521873024.

34. HOLCOMB, Daniel E.; BURLESON, Wayne P.; FU, Kevin. Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers. *IEEE Transactions on Computers* [online]. 2009, vol. 58, no. 9, pp. 1198–1210 [visited on 2022-04-13]. Available from DOI: 10.1109/TC.2008.212.

35. ANAGNOSTOPOULOS, Nikolaos Athanasios; ARUL, Tolga; SCHALLER, André; GABMEYER, Sebastian; KATZENBEISSER, Stefan. Low-Temperature Data Remanence Attacks Against Intrinsic SRAM PUFs. In: *2018 21st Euromicro Conference on Digital System Design (DSD)* [online]. 2018, pp. 581–585 [visited on 2022-04-17]. Available from DOI: 10.1109/DSD.2018.00102.

36. HELFMEIER, Clemens; BOIT, Christian; NEDOSPASOV, Dmitry; SEIFERT, Jean-Pierre. Cloning Physically Unclonable Functions. In: *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* [online]. 2013, pp. 1–6 [visited on 2022-04-08]. Available from DOI: 10.1109/HST.2013.6581556.

37. NEDOSPASOV, Dmitry; SEIFERT, Jean-Pierre; HELFMEIER, Clemens; BOIT, Christian. Invasive PUF Analysis. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography* [online]. 2013, pp. 30–38 [visited on 2022-04-08]. Available from DOI: 10.1109/FDTC.2013.19.

38. SHIFMAN, Yizhak; MILLER, Avi; KEREN, Osnat; WEIZMANN, Yoav; SHOR, Joseph. A Method to Improve Reliability in a 65-nm SRAM PUF Array. *IEEE Solid-State Circuits Letters* [online]. 2018, vol. 1, no. 6, pp. 138–141 [visited on 2022-04-13]. Available from DOI: 10.1109/LSSC.2018.2879216.

39. LIU, Muqing; ZHOU, Chen; TANG, Qianying; PARHI, Keshab K.; KIM, Chris H. A data remanence based approach to generate 100% stable keys from an SRAM physical unclonable function. In: *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)* [online]. 2017, pp. 1–6 [visited on 2022-04-13]. Available from DOI: 10.1109/ISLPED.2017.8009192.

40. ROELKE, Alec; STAN, Mircea R. Controlling the Reliability of SRAM PUFs With Directed NBTI Aging and Recovery. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* [online]. 2018, vol. 26, no. 10, pp. 2016–2026 [visited on 2022-04-13]. Available from DOI: 10.1109/TVLSI.2018.2836154.

41. MAES, Roel; LEEST, Vincent van der. Countering the effects of silicon aging on SRAM PUFs. In: *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* [online]. 2014, pp. 148–153 [visited on 2022-04-13]. Available from DOI: 10.1109/HST.2014.6855586.

42. DEUTSCHMANN, Martin; IRISKIC, Lejla; LATTACHER, Sandra-Lisa; MÜNZER, Mario; STORNIG, Felix; TOMASHCHUK, Oleksandr. Research on the Applications of Physically Unclonable Functions within the Internet of Things. In: [online]. 2018 [visited on 2022-04-19]. Available from DOI: 10.5281/zenodo.3606552.

43. ARCENEGUI, Javier; ARJONA, Rosario; BATURONE, Iluminada. Secure Management of IoT Devices Based on Blockchain Non-fungible Tokens and Physical Unclonable Functions. In: *Applied Cryptography and Network Security Workshops* [online]. Springer International Publishing, 2020, pp. 24–40 [visited on 2022-04-19]. ISBN 978-3-030-61638-0. Available from DOI: 10.1007/978-3-030-61638-0_2.

44. ESPRESSIF SYSTEMS CO., LTD. *ESP-IDF Programming Guide* [online]. 2022-01 [visited on 2022-04-14]. Available from: https://docs.espressif.com/projects/esp-idf/en/v4.4/esp32/esp-idf-en-v4.4-esp32.pdf.

45. ESPRESSIF SYSTEMS CO., LTD. *ESP32 Technical Reference Manual* [online]. 2021 [visited on 2022-04-14]. Available from: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

46. SOS ELECTRONIC S.R.O. *ESP32-WROOM-32U (ESP32-WROOM-32U-N16)* [online]. 2018 [visited on 2022-04-14]. Available from: https://cdn.sos.sk/productdata/6e/92/33234709/esp32-wroom-32u.jpg.

47. KRENT, Brian. *ESP32 Chip Function Block Diagram* [online]. 2018 [visited on 2022-04-14]. Available from: https://commons.wikimedia.org/w/index.php?title=File:Espressif_ESP32_Chip_Function_Block_Diagram.svg.

48. BINDER GMBH. *Data Sheet Model MK 56* [online]. 2021-12 [visited on 2022-04-16]. Available from: https://www.binder-world.com/en/content/download/122303/4169888/file/Data%20Sheet%20Model%20MK%20056%20en.pdf.

49. KOCANDA, Piotr; KOS, Andrzej. Static and dynamic energy losses vs. temperature in different CMOS technologies. In: *2015 22nd International Conference Mixed Design of Integrated Circuits Systems (MIXDES)* [online]. 2015, pp. 446–449 [visited on 2022-04-17]. Available from DOI: 10.1109/MIXDES.2015.7208560.

50. PUCHINGER, Sven; MUEELICH, Sven; BOSSERT, Martin; HILLER, Matthias; SIGL, Georg. On Error Correction for Physical Unclonable Functions. In: *SCC 2015; 10th International ITG Conference on Systems, Communications and Coding* [online]. 2015 [visited on 2022-04-21]. Available from: https://ieeexplore.ieee.org/document/7052113.

51. DODIS, Yevgeniy; OSTROVSKY, Rafail; REYZIN, Leonid; SMITH, Adam. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM Journal on Computing* [online]. 2008, vol. 38, no. 1, pp. 97–139 [visited on 2022-04-21]. Available from DOI: 10.1137/060651380.

52. HÁNOVÁ, Gabriela. *Výběr bitů pro SRAM PUF levného mikrokontroléru* [online]. Faculty of Information Technology, Czech Technical University in Prague, 2020 [visited on 2022-04-21]. Available from: https://dspace.cvut.cz/handle/10467/89959. MA thesis.

53. BATURONE, Iluminada; PRADA-DELGADO, Miguel A.; EIROA, Susana. Improved Generation of Identifiers, Secret Keys, and Random Numbers From SRAMs. *IEEE Transactions on Information Forensics and Security* [online]. 2015, vol. 10, no. 12, pp. 2653–2668 [visited on 2022-04-21]. Available from DOI: 10.1109/TIFS.2015.2471279.

Contents of the enclosed media

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF