Politecnico di Torino

Cybersecurity for Embedded Systems
01UDNOV

Master's Degree in Computer Engineering

Design and Development of a RAM-based PUF

Project Report

Candidates:
Zissis Tabouras (s284685)
Elena Roncolino (s304719)
Stefano Palmieri (s281677)

Referents:
Prof. Paolo Prinetto
Dr. Matteo Fornero
Dr. Vahid Eftekhari

Contents

List of Figures

List of Tables

Abstract

This is the space reserved for the abstract of your report. The abstract is a summary of the report, so it is a good idea to write after all other chapters. The abstract for a thesis at PoliTO must be shorter than 3500 chars, try to be compliant with this rule (no problem for an abstract that is a lot shorter than 3500 chars, since this is not a thesis). Use short sentences, do not use over-complicated words. Try to be as clear as possible, do not make logical leaps in the text. Read your abstract several times and check if there is a logical connection from the beginning to the end. The abstract is supposed to draw the attention of the reader, your goal is to write an abstract that makes the reader wanting to read the entire report. Do not go too far into details; if you want to provide data, do it, but express it in a simple way (e.g., a single percentage in a sentence): do not bore the reader with data that he or she cannot understand yet. Organize the abstract into paragraphs: the paragraphs are always 3 to 5 lines long. In LaTeXsource file, go new line twice to start a new paragraph in the PDF. Do not use to go new line, just press Enter. In the PDF, there will be no gap line, but the text will go new line and a Tab will be inserted. This is the correct way to indent a paragraph, please do not change it. Do not put words in **bold** here: for emphasis, use *italic*. Do not use citations here: they are not allowed in the abstract. Footnotes and links are not allowed as well. DO NOT EVER USE ENGLISH SHORT FORMS (i.e., isn't, aren't, don't, etc.). Take a look at the following links about how to write an Abstract:

- `https://writing.wisc.edu/handbook/assignments/writing-an-abstract-for-your-research-paper/`
- `https://www.anu.edu.au/students/academic-skills/research-writing/journal-article-writing/writing-an-abstract`

Search on Google if you need more info.

CHAPTER 1

Introduction

In the last years, the number of small electronic devices that can be connected with big computational units grew exponentially. Embedded systems play a crucial role in fueling the growth of the Internet-of-Things (Iot) in the most diverse domains, such as health care, home automation and transportation. By the end of 2022 the number of IoT devices connected to the Internet is expected to reach the astonishing number of 14.4 billions [1]. The ubiquitousness of such devices coupled with their ability to access potentially sensitive/confidential information has given rise to security and privacy concerns. An additional challenge is the growing number of counterfeit components in these devices, resulting in serious reliability and financial implications.

Physical unclonable functions (PUFs) are a promising security primitive to help address these concerns. PUFs extract secrets from physical characteristics of integrated circuits (ICs) [2] and therefore require minimal or no additional hardware for their operation and are therefore cheaper than other solutions. The instance-specific nature of the secret provide a mean to to uniquely identify and authenticate each device based on a challenge-response mechanism [3].

The aim of this project is to design and develop a RAM based PUF for the SEcube^{TM}, a single-chip easily integratable device capable of hiding significant complexity behind a set of simple high-level APIs [4].

The remainder of the document is organized as follows:

In Chapter 2, a brief background and state of the art of this topic is provided;
In Chapter 3, an implementation overview is presented;
In Chapter 4, implementation details are described;
In Chapter 5, results are listed;
In Chapter 6, conclusions and final observations are presented.
Appendix A describes how a demo of the implementation can be run.
Appendix B describes the APIs created for this project.

CHAPTER 2

Background

2.1 State of the art of embedded systems security approach

The current best practice for providing a secure memory or authentication source in mobile systems is to place a secret key in nonvolatile electrically erasable programmable read-only memory (EEPROM) or battery-backed static random-access memory (SRAM) and use hardware cryptographic operations such as digital signature or encryption. Nonetheless, this approach is expensive both in terms of design and of power consumption. In addition, invasive attack mechanisms make such nonvolatile memory vulnerable. Protection agains such attacks is therefore needed and it requires the use of active tamper detection/prevention circuitry which must be continually powered [2].

2.2 Physical unclonable functions

Physical unclonable functions (PUFs) are innovative primitives that derive secrets from complex physical characterstics of the ICs rather that storing the secret in digital memory. Because the PUF taps into the random variation during an IC fabrication process, the secret is extremely difficult to predict or extract. PUFs generate volatile secrets that only exist in a digital form when a chip is powered on and running. This requires the adversary to mount the attack while the IC is running and using the secret, which is significantly harder that discovering non-volatile keys. An invasine attack must measure the PUF delays without altering them or triggering sensing wires that clear out the registers [5].

The concept of PUFs is based on the idea that even though the mask and manufacturing process is the same during the creation of the same type of IC, each IC is actually slightly different due to normal manufacturing variability. PUFs leverage this variability to derive the silicon ”biometric”, a ”secret” information that is unique to the chip. This implies that no two identical chips can be manufactured. Although the use of PUFs is a relatively new technology, it should be noted that the concepts of unclonability and uniqueness of ojects have been extensively used in the past [2].

2.2.1 Types of PUFs

Most of the currently used PUFs fall into two categories:

- strong PUFs, mainly used for authentication, and
- weak PUFs, primarily used for key storage.

A PUF can be modeled as a black-box challenge-response system: an input challenge c is passed to a PUF which returns a response $r = f(c)$, where $f(\cdot)$ describes the input/output relations of the PUF. The black-box model is appropriate to describe PUFs since input parameters of $f(\cdot)$ are hidden from the user since they represent the interfan manufacturing variability that PUFs use to generate unique challenge-response sets.

The fundamental difference between weak and strong PUFs is the domain of $f(\cdot)$, i.e., the number of unique challenges c that the PUF can process. Weak PUFs can only support a small number of challenges (in some cases just a single challenge). On the contrary, a strong PUF can support a large enough number of challenges such that trying to determine/measure all challenge/response pairs (CRPs) within a limited timeframe is unfeasible.

Both weak and strong PUFs rely on analog physical properties of the fabricated circuit to derive secret information and therefore have noise and variability associated with them. For this reason, modern PUFs designs employ multiple error-correction techniques to mitigate the noise and improve realiability.

Examples of strong PUFs include optical and arbiter PUFs, while ring-oscillator and SRAM PUFs are example of weak ones. [2]

2.2.2 SRAM PUF

SRAM PUFs exploit the positive feedback loop in a SRAM. A SRAM has two stable states (used to store a 1 or a 0), and positive feedback to force the cell into one of these two state and to prevent an accidental state transition.

Figure 2.1 shows a common six-transistor configuration of an SRAM consisting of cross-coupled CMOS inverters (M_1-M_4) and access transistors (M_5-M_6).

Theoretically, when a device with a SRAM is powered on and no write operation is performed, the SRAM cell exists in a metastable state where the feedback pushing the cell toward the "1" state equals the feedback pushing the cell toward the "0" state, thereby keeping the cell indefinitely in this metastable state. However, in actual implementations one feedback loop is always slightly stronger than the other due to small transistor threshold mismatches resulting from process variation. This means that the cell at start up relaxes into either the "1" or "0" state. The final state of the cell depends on the difference between two feeback loops and it is therefore not strongly impacted by temperature or power supply fluctuations. Nonetheless, if the two feedback loops are sufficiently close then random noise or other small environmental fluctuations can result in an output bit flip. Therefore, error correction of this output will be necessary. Error correction can be performed by using repeated measurement: since the relative strengths between the two feedback is relatively static, by measuring the outputs of the cell repeatedly one can assess the stability of a SRAM PUF bit and selectively use the most stable bits as the PUF output. [2]

2.3 SEcube^{TM}

The SEcube^{TM} (Secure Environment cube) Open Security Platform is an open source security-oriented hardware and software platform. It provides hardware and software holistic security focusing on common operational security concepts like groups and policies instead of classical security concepts such as cryptographic algorithms and keys [6]. The SEcube^{TM} is the smallest reconfigurable silicon that combines three main cores in a single-chip design. It embeds a low-power ARM Cortex-M4 processor, a flexible and fast Field-Programmable-Gate-Array (FPGA), and an EAL5+ certified Security Controller (SmartCard), as shown in Figure 2.2. This make the SEcube a secure environment since it is based on a modular software architecture where all functions are isolated [7].

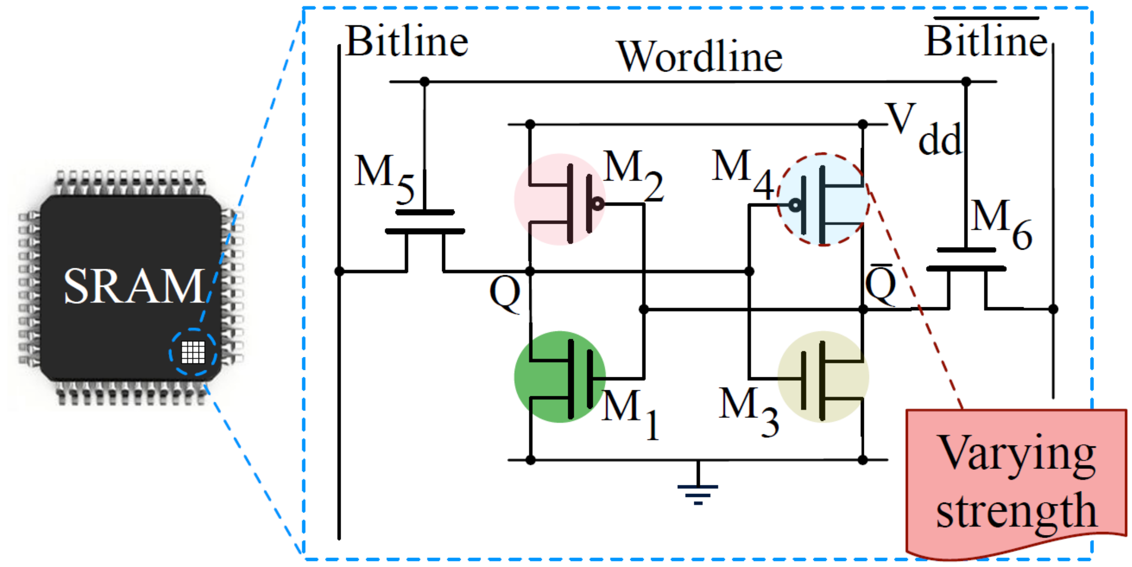


Figure 2.1: SRAM bit cells [3].

Figure 2.2: The three components of the SEcube$^{\text{TM}}$: the ARM Cortex-M4 processor, the FPGA and the EAL5+ SmartCard[6].

CHAPTER 3

Implementation Overview

As already said the aim of the project is to provide a secure PUF to recognize IoT devices, in order to avoid impersonation attacks.

The type of PUF implemented is a SRAM PUF (parlare un attimo dell' SRAM PUF se non e' stato fatto prima), this was implemented using a SECube device.

The Idea is that the first time an Host is connected to the SECube, it asks the device all the PUFs that it has collected during its starting phases. This challenge and response information has to be stored by the host in a cipher way:

$$data\_to\_store = H(response) \tag{3.1}$$

the challenge has to stay in plaintext.

In the future when the host has to establish a connection with the device, he is going to send to it a specific challenge, the device is going to answer with a response; then the host has to check the validity of the response, evaluating the digest and comparing with the one that he has in the storage file. Once a challenge-response is used, it has to be eliminated and not used anymore in the future; in this way it is possible to avoid replicant attacks.

The implementation of this project can be divided in two flow:

1. The first flow consists in the exchange of all the challenge and response information between host and device

2. The second flow consists in the challenge and response authentication of the device

Here will be explained only a general idea of the implementation and in the next chapter will be described deeply the functioning.

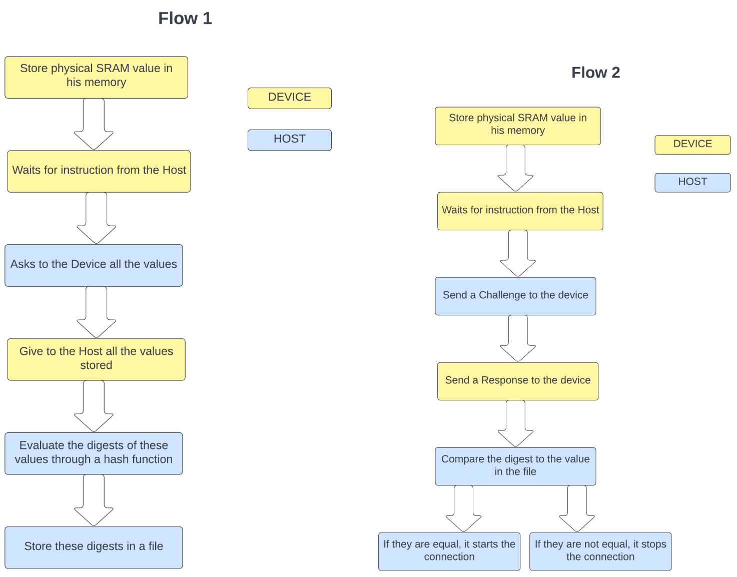

Flow 1
Store physical SRAM value in his memory
Waits for instruction from the Host
Asks to the Device all the values
Give to the Host all the values stored
Evaluate the digests of these values through a hash function
Store these digests in a file
DEVICE
HOST
Flow 2
Store physical SRAM value in his memory
Waits for instruction from the Host
Send a Challenge to the device
Send a Response to the device
Compare the digest to the value in the file
If they are equal, it starts the connection
If they are not equal, it stops the connection
DEVICE
HOST

Figure 3.1: Flow 1 and Flow 2

3.1 Device side

On the device side both of the flows have a common important operation that has to be done. This operation consists in taking the values present in the SRAM when this one is switched on and storing them in a secure place. Obviously this operation is the first operation that has always been done by the device as soon as it is switched on.

Secondly, if this is the first time that it communicates with this particular host it waits until the host asks it for all the value that it has just taken from the SRAM in order to implement the challenge-response authentication for the next time.

On the other hand, it waits until the host sends to it a particular challenge, this challenge will be the index of a determined response, so the device takes the response and sends them to the host. After that the device is authenticated and the can starts to communicate.

3.2 Host side

On the host side the type of work is a little bit different from the device side. When the connection is instantiated with the device for the first time it asks the device all the values needed to implement the challenge-response authentication; the Host is going to store the hash values of these datas in a file.

During all the next connections with the device, the Host sends a particular challenge to the device and waits for the respective response. After it receives it, It is going to evaluate the digest of this value, it gives it in input to the hash function and it compares it with the value that it has inside the file.

If the two digests are the same it means that the device is the correct one and so the connection can start; otherwise the host stops the connection.

CHAPTER 4

Implementation Details

This is where you explain what you have implemented and how you have implemented it. Place here all the details that you consider important, organize the chapter in sections and subsections to explain the development and your workflow.
Given the self-explicative title of the chapter, readers usually skip it. This is ok, because this entire chapter is simply meant to describe the details of your work so that people that are very interested (such as people who have to evaluate your work or people who have to build something more complex starting from what you did) can fully understand what you developed or implemented.
Don't worry about placing too many details in this chapter, the only essential thing is that you keep everything tidy, without mixing too much information (so make use of sections, subsections, lists, etc.). As usual, pictures are helpful.

4.1 Device side

The operations that the device must be able to perform are two:

- Read the PUFs from the RAM and store them in the SRAM, to be done the moment the device is powered on.
- Given a challenge, compare the PUF found in the DB of PUFs and the one found in the SRAM.

The implementation of both functionalities is based on the provided code and changes to it have been made to fit our purpose.

4.1.1 Generation of PUFs

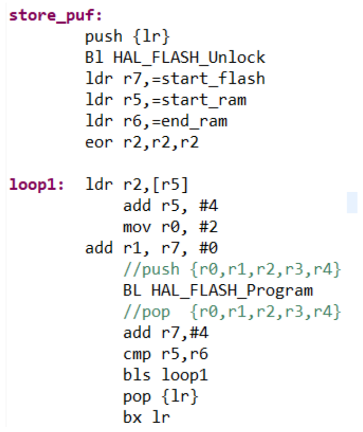
To make the retrieval of the PUFs possible two files had to be changed

1. *USBStick > Startup > startup_stm32f429nihx.s*
2. *USBStick > Src > Device > se3_dispatcher_core.c*

The .s file is the one in which the reset handler can be found and it will be executed before the main. Since the the content of the RAM will be overwritten the PUFs must be copied as soon as possible. For this reason this operation has been done before any memory initialization.

Using the reference manual for the MCU (STM32f429) we know the memory mapping of both the SRAM(0x020000000 - 0x02002FFFF) and flash memory(0x08000000 - 0x081FFFFF). Starting from that we scanned the SRAM and loaded the content to a part of the flash memory that will not compromise sensible data.

Since the the flash memory needs control registers to be set to guaranty access we called the HAL functions in the assembly code. More precisely the functions to unlock and program/write in memory. The code written to perform this operation is provided in figure 4.1



```
store_puf:
        push {lr}
        Bl HAL_FLASH_Unlock
        ldr r7,=start_flash
        ldr r5,=start_ram
        ldr r6,=end_ram
        eor r2,r2,r2

loop1:  ldr r2,[r5]
            add r5, #4
            mov r0, #2
        add r1, r7, #0
            //push {r0,r1,r2,r3,r4}
            BL HAL_FLASH_Program
            //pop  {r0,r1,r2,r3,r4}
            add r7,#4
            cmp r5,r6
            bls loop1
            pop {lr}
            bx lr
```

Figure 4.1: Assembly code to store PUFs into the flash memory.

At the end of the execution of this code we expect a number of PUFs to be stored in the flash memory which will be accessible once we enter the main and enter the execution loop where we can call our functions and access the content of the SRAM

The second modification that has to be made is the implementation of the function itself in the scheduler.

In the *se3_dispatcher_core.c* we implemented to functions To make the retrieval of the PUFs possible two files had to be changed

- *puf_retreive*()
- *puf_challenge*()

These two functions are the ones that will be called when the appropriate command will be sent from the host towards the device. More about the transmission of such message in section 4.2.

To make this possible it is necessary to define these commands in the $se3_dispatcher_core.h$ which must reflect the number associated also to the host side for the same command.

puf_retreive()

This function has the purpose to read the flash memory the locations where the PUFs have been stored and send them to the host to be managed accordingly. In the end it will provide a pointer to uint8. Considering that each PUF has a size of 32 bits it means that the size of the array will be four times the number of PUFs.

puf_challenge()

4.2 Host side

CHAPTER 5

Results

In this chapter we expect you to list and explain all the results that you have achieved. Pictures can be useful to explain the results. Think about this chapter as something similar to the demo of the oral presentation. You can also include pictures about use-cases (you can also decide to add use cases to the high level overview chapter).

5.1 Known Issues

One many issue of this kind of approach could be that there is not the possibility to avoid a Man-in-the-Middle, in order to avoid that a man can steal information from this kind of information it is necessary to encrypt the communication. It is important to say that the type of encryption and the necessity to encrypt or not depend from the type of device and by the level of sensibility of the datas.

5.2 Future Work

Many are the implementations that can be done on this project. The main one is to evaluate and store in a secure place the hash value of the file containing the challenge-response. This kind of implementation can be used in order to ensure the integrity of the challenge-response of a particular device. The idea consists in evaluating the hash value of the file before taking information from it and comparing it with the digest that we store in another place. If the value is the same it means that the file is not corrupted.

CHAPTER 6

Conclusions

To conclude, with this project it has been shown how a sram PUF works and how to implement it. In particular at the beginning It has been presented the problem present in these years, and why this kind of solution is important.

Subsequently has been shown some different kinds of PUF and how to approach the implementation of one of them....

It has been shown how to increase the security of the implementation with some simple precautions as a hash function that can help to increase integrity... .....

In the end it was shown some results of this approach....

To continue———————————————-

Bibliography

[1] IoT.Business.News, *State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally*, 2022, `https://iotbusinessnews.com/2022/05/19/70343-state-of-iot-2022-number-of-connected-iot-devices-growing-18-to-14-4-billion-globally/` [Online; Accessed 2022, 29 July].

[2] C. Herder, M. Yu, F. Koushanfar and S. Devadas, *Physical Unclonable Functions and Applications: A Tutorial*, in *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126-1141, Aug. 2014, doi: 10.1109/JPROC.2014.2320516.

[3] S. Sutar, A. Raha, and V. Raghunathan, *Memory-based Combination PUFs for Device Authentication in Embedded Systems*, for the School of Electrical and Computer Engineering, Purdue University, 5 December 2017, arXiv:1712.01611v1

[4] A. Varriale, E. I. Vatajelu, G. Di Natale, P. Prinetto, P. Trotta and T. Margaria, *SEcube™: An open-source security platform in a single SoC*, 2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2016, pp. 1-6, doi: 10.1109/DTIS.2016.7483810.

[5] G. E. Suh and S. Devadas, *Physical Unclonable Functions for Device Authentication and Secret Key Generation*, 2007 44th ACM/IEEE Design Automation Conference, 2007, pp. 9-14.

[6] M. Fornero, N. Maunero, P. Printetto, G. Roascio, A. Varriale, *SEcube^{TM} Open Security Platform, Introduction.* Released October 2021. `https://github.com/SEcube-Project/SEcube-SDK`.

[7] *What is SEcube^{TM}*, `https://www.secube.blu5group.com/`. [Online; Accessed 2022, 05 August].

[8] Donald E. Knuth (1986) *The TEX Book*, Addison-Wesley Professional.

[9] Leslie Lamport (1994) *LATEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.

APPENDIX A

User Manual

A.1 SEcube™ Software Development Kit (version 1.5.2)

A.2 Licence

All SEcube releases published on this website are Open Source - GPL 3.0 and are developed by the Academia Community.

A.3 Terms of use

By downloading the software from this page, you agree to the specified terms.

The software is provided to you "as is" and we make no express or implied warranties whatsoever with respect to its functionality, operability, or use, including, without limitation, any implied warranties of merchantability, fitness for a particular purpose, or infringement. We expressly disclaim any liability whatsoever for any direct, indirect, consequential, incidental or special damages, including, without limitation, loss revenues, lost profits, losses resulting from business interruption or loss of data, regardless of the form of action or legal thereunder which the liability may be asserted, even if advised of the possibility likelihood of such damages.

A.4 PUF

This is a project made for the course of Cybersecurity for embedded systems of the Masters on embedded system in Politecnico di Torino.

The purpose of this project is to extract the RAM PUFs from the memory of the SEcube storing them on a DB, in this case a simple txt file, and perform also a challenge on the board.

A.5 Instructions to run the project

A.5.1 Import the project

Instructions on how to import the project are the same as the ones for the original project provided in the wiki

A.5.2 Run the project

the steps to run the project are the following

- Erase the flash memory of the SEcube

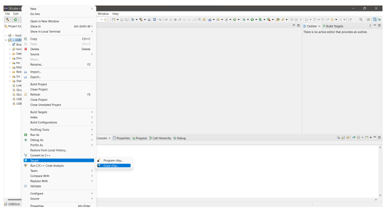


Figure A.1: Erase flash

- Flash the SEcube

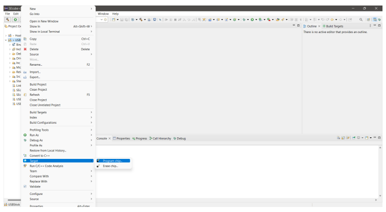


Figure A.2: Chip flash

- Run on the host the "puf_db_init.cpp"

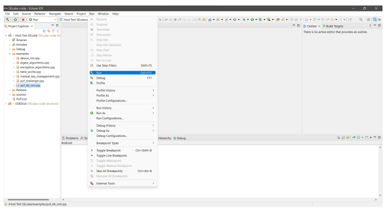


Figure A.3: Puf db init

- Run on the host the "puf_challenge.cpp"

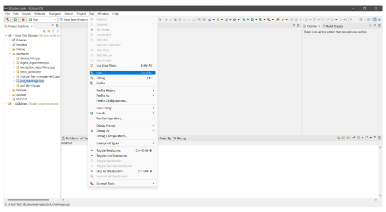


Figure A.4: Puf challenge

APPENDIX B

API

If you developed some source code that is supposed to be used by other software in order to perform some action, it is very likely that you have implemented an API. Use this appendix to describe each function of the API (prototype, parameters, returned values, purpose of the function, etc).