Open-Source Software-Based SRAM-PUF for Secure Data and Key Storage Using Off-The-Shelf SRAM

Ade Setyawan Sajim

Delft University of Technology

Open-Source Software-Based SRAM-PUF for Secure Data and Key Storage Using Off-The-Shelf SRAM

Master's Thesis in MSc Computer Engineering

Parallel and Distributed Systems Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Ade Setyawan Sajim

9th May 2018

Author
Ade Setyawan Sajim

Title
Open-Source Software-Based SRAM-PUF for Secure Data and Key Storage Using Off-The-Shelf SRAM

MSc presentation
17th May 2018

Graduation Committee

Dr. Ir. Johan Pouwelse	Delft University of Technology
Dr. Stephan Wong	Delft University of Technology
Dr. J.S. Rellermeyer	Delft University of Technology
Ir. Haji Akhundov	Delft University of Technology

Abstract

SRAM PUF has a potential to become the main player in hardware security. Unfortunately, currently available solutions are usually locked to specific entities, such as companies or universities. Here, we introduce the first open source project to develop software-based SRAM PUF technology using off-the-shelf SRAM. We also present testing results on two off-the-shelf SRAMs quality to be a PUF component; Microchip 23LC1024 and Cypress CY62256NLL. Testing on two bit-selection algorithms (data remanence analysis and neighbor analysis) are also performed. Based on the testing results, we introduce a PUF enrollment scheme using data remanence analysis as the bit selection algorithm which will locate the location of the stable bits and SRAM Cypress CY62256NLL as the off-the-shelf SRAM. Moreover, we also propose a procedure to develop SRAM PUF-based applications using any off-the-shelf SRAM. The procedure consists of three main steps; test the off-the-shelf SRAM quality to be a PUF component, create a PUF-generated key using enrollment-reconstruction mechanism, and develop any PUF-based application utilizing the PUF-generated key. In addition, an idea to create numerous CRPs using SRAM PUF is also proposed here. Using a collection of stable bits as a challenge, the stable bits are permutated among themselves to create a challenge which has a numerous number of possibilities. Furthermore, we also present a secure data and key storage scheme using SRAM PUF. The proposed scheme is influenced by multi-factor authentication. Using a combination of a PUF-generated key and user's password, a derived key is produced and utilized as the final key to protect user's data or/and user's key. As the grand concluding experiment of this thesis, we present a demo of storing a private key of Bitcoin. We shows that the Bitcoin key will not be reconstructed successfully if user's password is incorrect or the SRAM is not similar with the one that use to encrypt the Bitcoin key.

Preface

One of the main reasons why I am attracted to the field of computer science and engineering is the opportunity to develop useful products for society. An example which reflects this possibility is the open source project. Numerous people from around the world and different backgrounds are delved together to solve many issues continuously.

Sadly, when I start looking for a thesis topic, I realized that there is a subfield of computer security that still lacks public involvement. This field is called physically unclonable functions (PUF). On one hand, there are a lot of papers published and even some companies already selling some products based on this concept. On the other hand, there is no repository or project that initiate the public participation and provide working codes to get into a real product on this topic. There is some PUF-related public repository but it only offers PUF simulation. This may be also one of the reasons why PUF is not popular for common people even though it promised an excellent security feature. Based on this reason, I decided to delve into this problem and work on a thesis project which aims to provide the first open source PUF platform. In addition, due to the rise of self-sovereign identity concept, I also designed a secure data and key storage scheme based on the constructed PUF platform which will be helpful in addressing the problem of self-sovereign identity and keeping the secret key.

This thesis would not be going well without many support from various people. Johan Pouwelse, as my supervisor, I am really gladful for your guidance and enthusiasm in the last nine months. Stephan Wong, your suggestions and critics on the thesis ideas and report are invaluable. Jan Rellermeyer, thank you for being one of the graduation committee. Haji Akhundov, I am delighted to have many valuable feedbacks from you during our discussions. LPDP Indonesia Endowment Fund for Education Organizer, thanks for granting me financial support for my master degree. Last, I would like to thank my parents for the infinite love and support they have given me.

Ade Setyawan Sajim

Delft, The Netherlands
9th May 2018

Contents

Chapter 1

Introduction

This chapter starts by presenting background and motivation on doing this thesis project. Afterwards, the problem statement and thesis' goals will be explained. Explanations on our contributions to the state-of-the-art will continue this chapter. This chapter will be closed by description of thesis' outline.

1.1 Need for Self-Sovereign Identity

Hardly anyone can live without having their identity. Identity is the one that defines who we are, something which helps to describe the uniqueness of everyone. Identity's role in our daily life is unquestionable. Society requires identity systems to enable identity-requiring transactions at scale, allowing procedures that require the formal asking and answering of identity queries in place, allowing millions of transactions to occur. In modern society, identity is commonly related to social security cards, driver's licenses, and other state-issued credentials. Centralized controlled by the government is the definition of these elements.

Along with the rise of the digital age, identity also redefines itself. Identity in the digital world, can be referred as digital identity, is split into multiple domains. Our Facebook identity does not correlate directly to our Twitter identity or to most other domains. Digital identities are scattered, vary from one Internet domain to another. Scattered identities which locked to multiple entities leads to a problem where users are helpless in front of an authority who can deny their identity or even confirm a false identity. This phenomenon ignites a problem where users are not in control of their identity. There is no clear construction and agreement on how to build the digital identity which usable across platforms. This is an unfortunate since lack of digital identity also limits the development and delivery of efficient, secure, digital-based economy and society [1]. The failure to solved the digital identity problem issue even looks a bit strange since we already have *public key cryptography* since 1984, introduced by Chaum [2], which enable secure communication between parties without the hassle of key distribution problem and also provide valid digital signatures. Using public key cryptography, anyone who wants to send any message to a recipient needs to encrypt their message using the recipient's public key. Afterwards, the recipient can read the message after decrypting the received message using its private key.

To fix the scattered identity issue, a solution was proposed: one should be able to store their encrypted data in their own devices. To use the data, a service has to ask the data owner for the private key which will be used to decrypt the data. Using this concept, everyone has to keep their secret keys secure and solely in their possession. A centralized storage of private keys is out of question since it will be a honeypot for cyber attacks. Simply put, keeping secret keys secure is the cardinal problem to solve here.

All problems mentioned before lead to a thinking: identity and secure key storage need to be solved in a decentralized manner. In 2012, a new concept called *self-sovereign identity* (SSI) arose [3]. Self-sovereign identity is a decentralized identity concept which capable of authenticating statements, without any central organization, point-of-failure or any possibility of data tracking [4]. Self-sovereign identity will be able to give users full control over their identity. In simple words, users can store their identity data on their devices, and decide whether to give access to anyone who is willing to use it or not. In addition, there will be no need for a centralized storage since each user database is distributed among themselves. A high possibility to get this concept popular is also present with the introduction of the European Union General Data Protection Regulation [5].

At the end of 2017, Johan Pouwelse and Martijn de Vos proposed an SSI design which focused on data protection [4]. Data protection itself is related to securing data against unauthorized access [6]. Their proposal is described by a concept where the user data are encrypted and never leave the device/domain. Any operation which requires the data, such as authentication, will require symmetric encryption on the encrypted data. This encrypted data should be securely protected and the domain should be trustworthy. In addition, the key used to encrypt and decrypt data should be kept securely.

The most common way to store the key is by using a *non volatile memory* (NVM). NVM is a type of computer memory that keep intact its information without requiring a continuous power supply. An example of a product which uses NVM to store the key is a debit card where it uses its chip to store information. Unfortunately, this NVM is prone to physical attack. Since the key is permanently stored in the memory, an attacker can use some technique to clone the memory, such as *microprobing* [7]. An attacker may also use a side channel information to retrieve any information about the key. This attack can be even worse if someone that knows the system design is involved. Due to this problem, more secure, tamper-evident, tamper-proof solutions need to be presented.

1.2 Rise of PUF as a Security Solution

In 2001, Physical Unclonable Function (PUF) comes in handy as an inexpensive and yet effective security solution to overcome the mentioned problem before by a different way of generating and processing secret keys in security hardware. It was introduced by Pappu [8]. Unlike cryptographic algorithm security which usually relies on a hard-to-solve mathematical problem, PUF idea stems from using hardware features designed to utilize the physical random nanoscale disarray phenomena [9]. These disarray phenomena can be used as a derivation of keys without having to keep any security-critical information explicitly. This physical randomness is unclonable, even by the original manufacturer due to manufacturing process variations. Furthermore, since the secrets can only be produced when the PUF device is turned on, active manipulation of circuit structure will cause dysfunction of challenge-response mechanism and destroy the secret.

Related to self-sovereign identity concept, [4] present an idea to use PUF and biometric-based authentication to securely protect the data in the self-sovereign identity. Figure 1.1 shows the detailed technology stack in their trust creation proposal on how to build trust in the blockchain era.

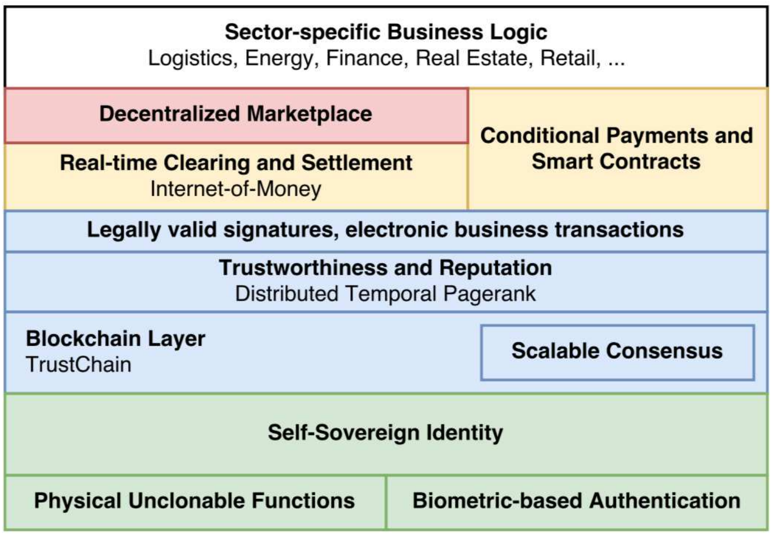

Sector-specific Business Logic
Logistics, Energy, Finance, Real Estate, Retail, ...
Decentralized Marketplace
Real-time Clearing and Settlement
Internet-of-Money
Conditional Payments and Smart Contracts
Legally valid signatures, electronic business transactions
Trustworthiness and Reputation
Distributed Temporal Pagerank
Blockchain Layer
TrustChain
Scalable Consensus
Self-Sovereign Identity
Physical Unclonable Functions
Biometric-based Authentication

Figure 1.1: Detailed technology portfolio for trust creation in the blockchain age [4]. As shown in the bottom of this figure, Physical Unclonable Functions and biometric-based authentication are utilized to secure the self-sovereign identity.

An example of PUF type is SRAM PUF. SRAM, stands for *static random-access memory*, is a type of semiconductor memory that uses bistable latching circuitry (flip-flop) to store each bit. When a static RAM (SRAM) is turned on, the memory cells have undefined states [10]. The initialized values on the memory cells are also random and unique to each SRAM. Based on these properties, SRAM is considered as a reasonable candidate for PUF. The value of these bits itself is determined by the SRAM cell which consists of two cross-coupled inverters along with two access transistors. This concept was first introduced by Guajardo and Holcomb in 2007 [11]. In order for SRAM to be used as a cryptographic security key, SRAM PUFs need to have certain characteristics such as the key generated by every SRAM should be reliable and unique. Reliable means the generated key should always be consistent, while unique refers to there should be no correlation between one device and another. Unfortunately, SRAM PUF is also problematic since it contains noise in its bit value. To handle the noise, error correction code is usually utilized.

1.3 Problem Statement

Since introduced by Guajardo and Holcomb in 2007, there have been many innovations in SRAM PUF field. A simple patent search using patents.google.com with query 'sram; puf' results in 546 results [12]. The number of articles in scholar.goog le.com also exhibit a high occurrences, shown by a total of 2,120 articles (citations and patents are not included) [13]. Even though these facts indicate a promising future for this concept, one also should notice that current state-of-the-art in this field mostly consists of one-off prototypes or specific proprietary implementations. To get an SRAM PUF product from the market, one has to order a specific request from a company. For example, Intrinsic-ID, one of the main leaders in SRAM PUF technology, has a software-based solution which able to generate unique keys and identities for nearly all microcontrollers without a need for security-dedicated silicon [14]. Even though this solution exists and seems easy to use, unfortunately, they do not say specifically how much will it cost to use this solution. They also have another solution for SRAM PUF which is focused on hardware IP (and supporting software/firmware) to enable designers to implement PUFs within their design. This solution has a high possibility to obstruct a small company or a single user to use their solution since usually this type of product are intended to use with a specific contract. Similar to the software-based solution they offer, they also do not put the explicit price to use this product. An example of a product that uses this solution is FPGA Microsemi Polarfire [15].

The SRAM PUF field lacks an Arduino, Linux, or GCC type of open reference implementation. A quick lookup in Github shows that there is no extensive open source project related to SRAM PUF there. There are projects corresponding to PUF concepts, but most of them also only delve into a simulation. The communities seem to have not established a wide agreement on which approach yields the strongest security properties.

An additional issue that we would like to address is SRAM PUF's application. As mentioned in Section 1.1, the importance of securing key and user's data is getting higher, especially with the introduction of self-sovereign identity. There are already many SRAM PUF applications published, but sadly, there is no working project that tries to integrate SRAM PUF in self-sovereign identity concept. Most PUF applications are designed for authentication [16] [17] [18] [19] [20] [21] and generating cryptographic keys [18] [22].

Based on these facts, we believe the next challenge for this field is to discover a common approach. The field needs to move beyond isolated single-person projects and single-company approaches towards a mature and sharing ecosystem. The field SRAM PUF requires a single implementation which is continuously improved upon for many years to come and is supported by the majority of the academics and commercial parties. Furthermore, we also try to initiate an integration between PUF and self-sovereign identity by providing a scheme to protect user's data and key. This project will be useful in the process of self-sovereign identity development.

To understand our intention in this thesis better, this thesis' problem statement is presented here. The problem statement of this thesis is:

> *How to develop an open-source secure data and key storage scheme using off-the-shelf SRAM component and software-based SRAM PUF technology?*

Derived from the problem statement, there are two goals defined in this thesis. The first goal is to devise a secure data and key storage scheme based on SRAM PUF technology. The data and the key protected by the scheme has to be safe even though the PUF device is lost. Moreover, the scheme should work using off-the-shelf SRAM. This sub-goal leads us to another question, can we build SRAM PUF using off-the-shelf SRAM? If it is possible, what characteristics need to be fulfilled by off-the-shelf SRAM to be eligible as a PUF candidate? In addition, the constructed SRAM PUF has to work without any hardware design, or in other words, software-based construction. The secure data and key storage functions inside the scheme will be helpful in addressing the problem of self-sovereign identity and keeping the secret key. The second goal is to create a sharing ecosystem for the evolution of our data and key storage scheme. The ecosystem should be easily accessed and understood to encourage the academics and commercial parties to use and develop the ecosystem together.

1.4 Contributions

In our work, we strongly believe in open source idea and communities involvement when developing a system. Combined with the problems and potential of SRAM PUF mentioned before, this thesis generates several additions into the state of the art of SRAM PUF knowledge. This thesis' contributions are explained below:

- *The first open source project on software-based SRAM PUF using off-the-shelf SRAM.* This software-based SRAM PUF project consists of Arduino and Python codes and can be found on a Github repository [23]. It provides the off-the-shelf SRAM testing, enrollment and reconstruction mechanism which can be utilized to develop other applications. The testing part can be utilized to check whether an SRAM is capable to be a PUF root-of-trust or not. The enrollment stage will generate the helper data and the challenge which stored on a microSD connected to Arduino. The reconstruction part can generate a PUF-generated key based on the challenge and the helper data. In our construction, the selected off-the-shelf SRAM is Cypress CY62256NLL. We also tested another type of SRAM called Microchip 23LC1024 but we abandoned it due to insufficient results to be eligible as a PUF candidate. The enrollment stage also requires a bit selection algorithm called data remanence analysis. In the experiment part, there is another bit selection algorithm tested named neighbor analysis. This method is not selected due to worse performance than data remanence analysis.

- *Procedure to develop an SRAM PUF-based application using any off-the-shelf SRAM.* The procedure consists of three main steps. First, one should test the off-the-shelf SRAM quality to be a PUF component. If passed, the procedure continues to the next step which consists of enrollment and reconstruction mechanism which will be able to create a PUF-generated key. Last, using the PUF-generated key, one can develop any PUF-based application.
- *A scheme to enable secure data and key storage using off-the-shelf SRAM and software-based SRAM PUF.* This scheme is influenced by multi-factor authentication. Using a combination of the PUF-generated key and user's password, a derived key is produced and utilized as the final key to protecting user's data or/and user's key.
- *A concept to devise numerous CRPs using SRAM PUF.* One of SRAM PUF drawbacks is the limitation of possible challenge-response pairs. We propose to use a set of bit locations as the challenge since when using this concept, the number of possible pairs is the permutation of total bit locations over the required number of bit locations. The total possible CRPs using this concept is a significant large number which can be bigger than the total number of atom in earth.

1.5 Outline

After explaining a brief review of SRAM's potentials and problems, problem statement and our contributions in this chapter, Chapter 2 continues with an overview of security, cryptography, symmetric encryption, key derivation function and multi-factor authentication. Explanations of PUF and SRAM PUF are also presented in that chapter. Chapter 3 describes our proposed SRAM PUF development system, our idea on how to create numerous CRPs using SRAM PUF, and a scheme to enable secure data and key storage using SRAM PUF. Chapter 4 shows our implementation, experiments and results. Last chapter, Chapter 5, summarizes this thesis and also gives our view on possible improvements on this project.

Chapter 2

Related Work

This chapter examines some background theory related to security, cryptography, and PUF. A brief review of security is presented, followed by explanations on symmetric cryptography, key derivation function and multi-factor authentication. Then, theories related to PUF and SRAM PUF are described, continued by evaluation on some PUF-based applications. We also present previous publications which related to SRAM PUF built using off-the-shelf SRAM.

2.1 Security Requirements and Cryptography

A perfect and 100% secure system is the holy grail of all computing system. Unfortunately, such thing does not exist. The best way to achieve that goal is by designing a system to be as secure as possible in a limited scope. To help defining a secure system, common security requirements are mentioned. According to [24], there are four elements on common security, which are:

- *Confidentiality*: a piece of information should be accessible only to an authorized user. For example, an encrypted data can only be decrypted by the secret key owner.
- *Authentication*: assurance of the sender of a message, date of origin, data content, time sent, data information, etc. are correctly identified.
- *Integrity*: any assets can only be modified by authorized subjects. For example, data should be kept intact during transmission.
- *Non-repudiation*: a subject should be prevented from denying previous actions. For example, a sender cannot deny the data which it sent.

One way to achieve these four security requirements is by using cryptography. In traditional definition, cryptography can be defined as the art of writing or solving codes [25]. But this definition is inaccurate to use nowadays because instead of depending on creativity and personal skill when constructing or breaking codes, the modern cryptography focuses their definition using science and mathematics. According to [26], modern cryptography can be defined as “the scientific study of techniques for securing digital information, transactions, and distributed computations.” The algorithm which uses cryptography as their main point is called cryptographic algorithm.

Since the birth of cryptography, its main concerned is usually related to securing communication which can be achieved by constructing *ciphers* to provide secret communication between parties involved. The construction of ciphers to ensure only authorized parties also can be called as encryption schemes. A secure encryption scheme should be able to provide indistinguishability on the produced ciphertext. The highest indistinguishability level that can be achieved by encryption scheme is *IND-CCA* (indistinguishability under chosen ciphertext attack). There are two types of encryption algorithm; symmetric and asymmetric encryption algorithm. Symmetric, also known as private key encryption or private key cryptography, requires the same key for encryption and decryption. Meanwhile, in the asymmetric algorithm (can be referred as public key encryption or public key cryptography), there are two keys utilized; private key and public key. A public key is utilized for encryption and a private key is used for decryption. One of the main advantages of symmetric encryption over asymmetric encryption is it requires less computational power which makes it suitable to use in embedded devices.

Besides encryption, another application of cryptography is for authenticating a message. A message can be proven its integrity by creating a *message authentication code (MAC).*

2.2 Symmetric Encryption

According to [26], symmetric encryption consists of three algorithms which are:

- *Gen*: key-generation algorithm
- *Enc*: encryption algorithm
- *Dec*: decryption algorithm

To illustrate this better, an example using two parties, Alice and Bob are given. Before using the encryption or decryption algorithm, both parties will agree on a shared secret key k. This phase can be referred as Gen. Afterwards, Alice can use the encryption algorithm (Enc) E_k using the shared secret key k on a message m which will generates a ciphertext c. This procedure can be noted as $c = E_k(m)$. Bob can read the message by using the decryption algorithm ((Dec)) Dec_k using the same shared secret key k. Decryption will result in the plaintext message m. This can be noted as $m = D_k(c)$.

There are many examples of symmetric encryption algorithms, such as RC2, DES, 3DES, RC6, Blowfish, and AES. AES algorithm will be explained below.

AES

AES, stands for Advanced Encryption Standard, is an encryption algorithm based on a substitution-permutation network and established by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES is an example of block ciphers (iteratively works on blocks of plaintext to produce blocks of ciphertext) [27]. The block size inside AES has a size of 128 bits, while the key size can be either 128, 192, or 256 bits. The key size itself describes the number of rounds which convert the plaintext into the ciphertext. If 128-bit key is used, there are 10 rounds utilized. 192-bit key leads to 12 rounds, while 14 rounds is used when 256-bit key is applied.

There are four major parts inside AES; *KeyExpansions*, *InitialRound*, *Rounds* and *FinalRound*. In KeyExpansions, the round keys are generated using Rijndael's key schedulebased on the AES key. Inside a normal round, there are four stages required to do; *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. SubBytes refers to a non-linear substitution procedure using a lookup table. ShiftRows means an act of shifting cyclically the last three rows of the state. MixColumns contains a mixing activity on the columns of the state. AddRoundKey involves a fusing process of each byte of the state with a block of the round key utilizing bitwise XOR operation. The difference between InitialRound, Rounds, and FinalRound is InitialRound only contain AddRoundKey, FinalRound does not has MixColumns inside, and Rounds just filled with those four stages.

An encryption can be done by following all these four parts. To convert ciphertext into the original plaintext, it is only required to apply a set of reverse rounds using the same encryption key.

2.2.1 Modes of Operation

Modes of operation refer to various ways to use a block cipher, like AES or DES. Below are five examples of modes of operation [27]:

- *ECB (Electronic Codebook)*: Plaintext is divided into multiple blocks, then encryption is done on each block separately.

- *CBC (Cipher Block Chaining)*: Plaintext is divided into multiple blocks, then encryption is done on the result of an XOR between a plaintext block and the previous ciphertext block.

- *OFB (Output Feedback)*: Plaintext is divided into multiple blocks. Then, it generates keystream of blocks by encrypting the previous block of key. A block of ciphertext is produced by XOR-ing a block of keystream with a block of plaintext.

- *CFB (Cipher Feedback)*: Plaintext is divided into multiple blocks, then a block of ciphertext is produced by XOR-ing the encryption of the previous block of ciphertext and the plaintext.
- *CTR (Counter))*: Plaintext is divided into multiple blocks. Later, it creates keystream of blocks by encrypting consecutive values of a "counter". A block of ciphertext is produced by XOR-ing a block of keystream with a block of plaintext.

From these five modes of operation, four of them requires an IV (initialization vector). The only one that does not need an IV is ECB mode.

2.2.2 Encrypt-then-MAC

The highest indistinguishability level that can be achieved using AES and five modes of operation mentioned before is *IND-CPA* (indistinguishability under chosen plaintext attack) [27]. IND-CPA can be achieved when using CBC, OFB, CFB, and CTR mode with a random IV. CTR mode can also achieve IND-CPA level when using nonce-based IV. To achieve IND-CCA level, a technique called *Encrypt-then-MAC* can be utilized. This technique will prevent an adversary to create any ciphertext without knowing the underlying key.

To use this procedure, two keys *K1* and *K2* need to be generated first. Afterwards, use *K1* to encrypt the plaintext P which result in C and *K2* for computing the MAC of the ciphertext C which result in H. During decryption, one should provide $\widetilde{K}1$ and $\widetilde{K}2$. The input ciphertext during decryption will be referred as $\widetilde{C}$. Next, ensure the MAC of the given ciphertext $\widetilde{C}$ using $\widetilde{K}2$ is similar with the stored MAC (H). Similar MAC means that the ciphertext C is not altered (same as $\widetilde{C}$) and the adversary use correct $K2$. If similar, give the result of the decryption of ciphertext $\widetilde{C}$ using $\widetilde{K}1$.

2.3 Key Derivation Function

Besides the encryption algorithm, a *key derivation function* (KDF) is one of the most utilized components of cryptographic applications. Its importance is due to its ability to convert a stable secret Z, usually contain sufficient amount of randomness but non-uniformly distributed, into one or more cryptographically strong secret keys $k \, \epsilon \, 0, 1^K$ where K is the length of the generated keys. Cryptographically strong itself refers to indistinguishability by reasonable computation from a random uniform string with similar length [28]. KDF can also be referred as a strong extractor. A popular example of KDF is a keyed cryptographic hash function (can be referred as *HMAC*, stands for hash-based message authentication code).

There are three elements which defined the cryptographic strength of the HMAC; the utilized hash function's cryptographic strength, the output size, and the key's size and quality. Currently, the latest cryptographic hash function standard published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS) is SHA-3 (introduced in 2015). SHA-3 (Secure Hash Algorithm 3) is a part of another cryptographic primitive family called Keccak [29]. Keccak is built on top of a method called *sponge construction*. Sponge construction itself is based on multiple layers of pseudorandom function where each layer capable of mapping variable-length input to variable-length output using fixed-length permutation (or transformation) and a padding rule [30].

There are three requirements need to be fulfilled as a secure cryptographic hash function: *preimage resistant, second preimage resistant, and collision resistant*. Preimage resistant means it should be hard to find a message with a given hash value. In second preimage resistant, if one message is provided, it should be hard to find another message with the same hash value. Last, collision resistant refers to difficultness to find two messages with the same hash value. HMAC built using SHA-3 with key length of 256 bits has collision resistance of 128 bits, preimage resistance of 256 bits, and second preimage resistant of 256 bits [31].

2.4 Multi-factor Authentication

As mentioned in Section 2.1, authentication refers to assuring any piece of information is correctly identified. Authentication can be done using any of these elements/factors; knowledge (a piece of information which only known by the user, e.g. password), possession (any object which only owned by the user, e.g. RFID card), or inherence (something which uniquely describe the user, e.g. fingerprint). If two or more elements are combined together for authentication, this leads to *multi-factor authentication*. To understand the security level among all possible combinations, Figure 2.1 is provided. The highest possible security level is when these three factors are combined together.

2.5 Physically Unclonable Function

A physically unclonable function (PUF) is an entity that utilizes manufacturing variability to produce a device-specific output. The idea to build PUF arise from the fact that even though the mask and manufacturing process is the same among different ICs, each IC is actually slightly different due to normal manufacturing variability [9]. PUFs leverage this variability to derive secret information that is unique to the chip. This secret can be referred

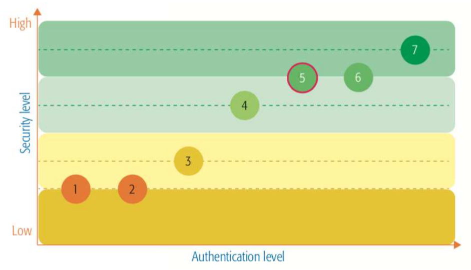


Figure 2.1: Authentication systems security levels: (1) knowledge; (2) possession; (3) knowledge + inherence; (4) inherence; (5) possession + inherence; (6) knowledge + inherence; (7) knowledge + possesion + inherence [32].

as a silicon biometric. In addition, due to the manufacturing variability that defines the secret, one cannot produce two identical chips (identic in nanoscopic scale), even with full knowledge of the chips design. PUF architectures exploit manufacturing variability in multiple ways. For example, one can utilize the effect of gate delay, the power-on state of SRAM, threshold voltages, and many other physical characteristics to derive the secret.

Due to this feature, PUFs are a promising innovative primitive that is used for authentication and secret key storage. Currently, the best practice for providing a secure memory or authentication source in such a mobile system is to place a secret key in a nonvolatile electrically erasable programmable read-only memory (EEPROM) or battery-backed static random-access memory (SRAM) and use hardware cryptographic operations such as digital signatures or encryption.

There are two main parts of PUF, physical part, and operational part. Physical part refers to a physical system that is very difficult to clone due to uncontrollable process variations during manufacturing. Operational part means a set of *challenges* (PUF input) C_i has to be available to which the system responds with a set of sufficiently different *responses* (PUF output) R_i. This combination of challenge and response is called *challenge-response-pair* (CRP).

$$R_i \leftarrow PUF(C_i) \tag{2.1}$$

The common application on using PUF usually requires two phases; the first phase is called *enrollment* and the second one is usually referred as *validation*. In enrollment, a number of CRPs are gathered from a PUF and then stored. In validation phase, a challenge from the stored CRPs is given to the PUF. Afterwards, the PUF response from this challenge is compared with the corresponding response from the database. The response is considered to be valid if there is a CRP from the stored CRPs related to this challenge and response. The validation phase can also be referred as *reconstruction* phase since this phase involves a reconstruction of a response given a challenge.

According to [9], to be qualified as PUF, a device should fulfill several characteristics below:

- *Reliable*: A response to the same challenge should be able to be reproduced over time and over a various range of conditions.
- *Unpredictable*: A response to a challenge on a PUF device should be unrelated to a response to another challenge from the same device or the same challenge from a different device.
- *Unclonable*: Challenge-response pairs mapping of a device should be unique and cannot be duplicated.
- *Physically Unbreakable*: Any physical attempts to maliciously modify the device will result in malfunction or permanent damage.

2.5.1 PUFs Classification

In this subsection, two subtypes of PUFs so-called "Weak PUFs" and "Strong PUFs" are presented. The explanations on both types can be found below [33]:

- Strong PUFs
  Strong PUFs can be recognized by possessing a tremendous number of CRPs which prevent an adversary to read all possible CRPs even if he has open access to the challenge-response interface. Anyone can freely give any challenge and read the response without affecting its security. In addition, even if he has a large subset of CRPs, he still cannot predict another yet unknown CRPs. Strong PUFs typically used for authentication.
- Weak PUFs
  Weak PUFs can be identified by having few CRPs. Unlike the strong PUFs, weak PUFs require an access-restricted to the challenge-response mechanism. This means that even if an adversary holds a possession of the PUF device, he cannot read the response from a challenge or give any challenge to the PUF device. Weak PUFs commonly used for key storage and key generation.

Besides the number of CRPs, PUFs can also be categorized based on their physical design. There are two major categories, extrinsic and intrinsic PUFs [34].

Extrinsic PUFs are built based on the explicitly introduced randomness in the system. Explicit randomness can be generated using various ways, e.g. use specific materials or size of particles, but the location and the distribution of this randomness cannot be controlled. There are two subcategories of extrinsic PUFs, non-electronic and analog electronic PUFs. Some examples in non-electronic PUFs are optical PUF, paper PUF, CD PUF, RF-DNA PUF, magnetic PUF, and acoustic PUF. Some design instances in analog electronic PUFs are VT PUF, power distribution PUF, coating PUF, and LC PUF.

In intrinsic, PUF component has to be available naturally during the manufacturing process. In addition, PUF and the measurement equipment should be fully integrated with intrinsic PUF. There are two subcategories in intrinsic PUFs, delay based and memory based PUFs. An example of delay based PUF is arbiter PUF. The main principle of arbiter PUF is by presenting a race condition on two different routes on a chip where the winner will be decided by an arbiter circuit [35]. As in memory based PUFs, some examples of this design are SRAM PUF, butterfly PUF and latch PUF. SRAM PUF utilized the random physical mismatch in the cell introduced by manufacturing variability which controls the power-up behavior (can be zero, one, or no preference) [35]. Butterfly PUF use the effect of cross coupling between two transparent data latches. Using the functionalities of the latches, an unsteady condition can be initiated after which the circuit resolves back to one of the two stable states [35]. In latch PUF, the concept is based on using two NOR gates which are cross-coupled. These gates will lead to a stable condition depending on the internal discrepancy between the electronic components.

2.5.2 Hamming Distances as an Identification Helper

As explained before, PUF main purpose is dedicated for identification, shown by having a device-specific output. In PUF, *hamming distance* is commonly used as a way to help defining this idea. Hamming distance itself is the number of positions at which the corresponding symbols are different on two equal length strings [36]. There are two types of hamming distance utilized, intra-chip and inter-chip hamming distance. Inter-chip hamming distance is the distance between two responses resulting from giving a similar challenge to two distinct PUF devices [35]. Intra-chip hamming distance refers to the difference between the two responses resulting from applying a challenge twice to a PUF device [37]. To ease the identification purpose, fractional hamming distance is also introduced. Fractional hamming distance is the number of differences between two strings divided by the length of the bit strings. In ideal PUFs, the intra-chip fractional hamming distance (HD_{intra}) is 0% and inter-chip fractional hamming distance (HD_{inter}) is 50%. The identification goal will not be achieved if there is an overlap between HD_{intra} and HD_{inter} [38]. Overlap will happen if the HD_{intra}

is too large and HD_{inter} is too small, e.g. HD_{intra} is 35% and HD_{inter} is 30%.

2.5.3 Helper Data Algorithms and Fuzzy Extractor

There are two issues if PUF raw responses are used as a key in cryptographic primitive. First, both weak and strong PUFs rely on analog physical properties of the fabricated circuit to derive secret information. Naturally, these analog properties have noise and variability associated with them. This can be a problem due to sensitivity of cryptographic functions on noises of their inputs. Another issue is the PUF raw responses usually are not uniformly distributed, which makes it unqualified as a cryptographically secure key. These two issues can be solved using *Helper Data Algorithm* (HDA). One can also refer Helper Data Algorithm as *fuzzy extractor* since both are capable of converting noisy information into keys usable for any cryptographic application [39].

Fuzzy extractor solves both issues mentioned before by using two phases, *information reconciliation* and *privacy amplification*. In information reconciliation phase, possible bit errors are corrected to form a robust bit string [40]. Information reconciliation is tightly related to error correction. In fact, a procedure to do information reconciliation based on error-correcting codes is called code-offset technique [41]. Using code-offset technique, one should be able to reconstruct a bit string *w* from a noisy version *w'* as long as the Hamming distance between *w* and *w'* is limited to *t* where t is the maximum error correcting capability of the error correcting codes. The second phase, privacy amplification, is a process to evolve this robust bit string into a full entropy key. Privacy amplification, also can be called as randomness extraction [42], can be done by utilizing two-way hash function.

Beside these two phases, fuzzy extractor also consists of two procedures, *generation* and *reproduction*. Generation is a probabilistic procedure which outputs an "extracted" string / key (secret) R and a string (public) *helper data* P on input fuzzy data w. Reproduction is a deterministic function capable of recovering secret key R from the string *helper data* P and any vector w' as long as the Hamming distance between *w*and *w'* is limited to *t* where t is the maximum error correcting capability of the fuzzy extractor.

In [43], Taniguchi et. al. illustrated the generation and reproduction procedure of fuzzy extractor on PUF which is shown in Figure 2.2. During generation phase, the secret key is produced by hashing the PUF response while helper data is generated by XOR-ing PUF response with encoding result of a random number. During reproduction stage, the secret key can be reconstructed by hashing the XOR result between helper data and encoding of the reproduced random number. The random number can be rebuild by decoding the XOR result between helper data and the PUF response.

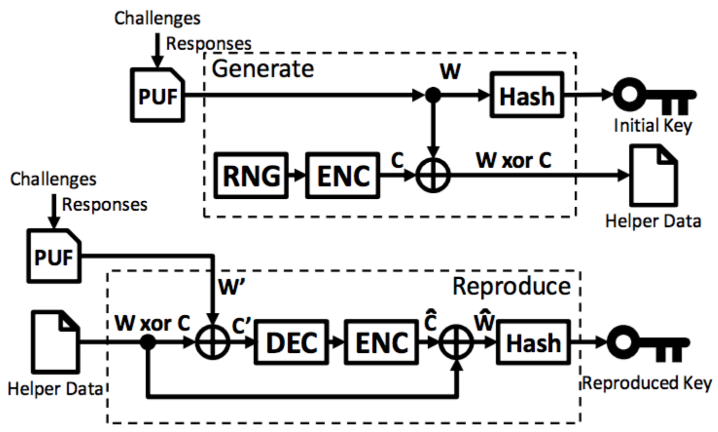


Figure 2.2: Generation and reproduction procedures of fuzzy extractor on PUF [43].

2.5.4 Error Correcting Codes

To handle noises occurred inside a PUF, error-correcting codes (ECC) is employed. Error-correcting codes are a class of schemes for encoding messages in an attempt to enable message recovery when there is noise introduced in the sending or receiving of the message [44]. ECC can be divided into two subcategories, hard-decision and soft-decision. Hard-decision works on a predetermined set of values (usually 0 or 1 in a binary code), while a soft-decision decoder may take inputs on a span of values in-between (usually refers to float value).

There are some well-known ECC, such as in hard-decision code, Reed-Solomon code and BCH code; while in soft-decision, Viterbi code and turbo code. Soft-decision code has an advantage over hard-decision code where it can process extra information which indicates the reliability of each input data point and used to form better estimates of the original data. But it has drawback where one should provide a probability function on the data (on SRAM, a probability function on each cell should be provided) to enable a good decoding result.

One of the popular hard-decision error correcting code is BCH codes. BCH, stands for *Bose-Chaudhuri-Hocquenghem*, codes are a family of cyclic error correcting codes which constructed using polynomials over a finite field and work in a binary field [44]. BCH codes are a very flexible set of codes in that within certain bounds there is a great amount of choice in code parameters and are relatively efficient in message length and error correction. The code parameters are as follows:

- q: The number of symbols used (e.g., in binary field, $q = 2$)

- m: The power to which to raise q to generate a Galois Field for the construction of the code.
- d: The minimum Hamming distance between distinct codewords.

These parameters lead to several derived parameters which are standard parameters of linear codes:

- n: The block length of the code; for our special case, $n = q * m{-}1$
- t: The number of errors that can be corrected, $d \geq 2t + 1$
- k: The number of message bits in a codeword, $k \geq n - mt$

Both BCH codes and Reed-Solomon codes have the capability to correct multiple errors. Reed-Solomon codes are also a flexible ECC and have similar parameters as BCH codes, e.g. n, k, d. Unlike BCH codes, Reed-Solomon codes can work in both binary and non-binary fields. Reed-Solomon codes also perform better in correcting burst errors while BCH codes are better at fixing random errors. BCH codes have an advantage where it requires less computing resource when working on the same parameter compared to Reed-Solomon codes.

2.6 SRAM PUF

SRAM PUF was first proposed by Guajardo and Holcomb in 2007. SRAM PUF uses existing SRAM blocks to generate chip-specific data. Normally, when using SRAM to store data, a positive feedback is given to force the cell into one of the two states (a '1' or a '0') available. Once it is there, the cell will be stable and prevented from transitioning out of this state accidentally.

SRAM can be used as a PUF by utilizing its start-up values. After powering-up the circuit, each cell stabilizes at a state which is defined by the mismatches between the involved transistors and provides one bit of output data. Since this mismatch determines the value of the power-up state of an SRAM cell, the power-up state of a cell will be biased towards 0 or 1 depends on the mismatch value. Since all SRAM cells have been affected by random process mismatches and non-identical, these start-up SRAM values can be utilized to generate a unique fingerprint [45].

2.6.1 Requirements for SRAM to be a PUF Component

To be eligible as a PUF component, an SRAM has to have stable outputs which means any noise has to have little effect on its start-up behavior (shown by the value of $\mathrm{HD}_{\mathrm{intra}}$). In addition, the distribution of 1's and 0's in the SRAM values ideally has to be equal (around 50:50) to ensure there is sufficient amount of randomness exist in the SRAM [46]. The distribution of 1's and 0's can also be referred as *hamming weight*. Moreover, the difference between responses from different chips given the same challenge should be large enough to show that each SRAM is unique (there should be no overlap between HD_{intra} and HD_{inter}).

2.6.2 SRAM Cell

SRAM uses its SRAM cells to store the binary information. The most common SRAM design is six-transistor (6-T) CMOS SRAM, shown in Figure 2.3. This design utilizes the concept of cross-coupled inverters, constructed by two inverters, each established by two transistors; inverter 1 by Q2 and Q6, inverter 2 by Q1 and Q5. Using this design means the input of an inverter is the output of the other and vice-versa, which also indicates that the output of one inverter is exactly the opposite of the other inverter [37]. Transistors Q3 and Q4, referred as the access transistors, are used as the entry gate to the cell every time a read or write operation will be performed. The bitline (BL), the compliment bitline (BLB) and the wordline (WL) are employed as an entry to the cell. In addition, an SRAM cell will lose its state shortly after power down [34].

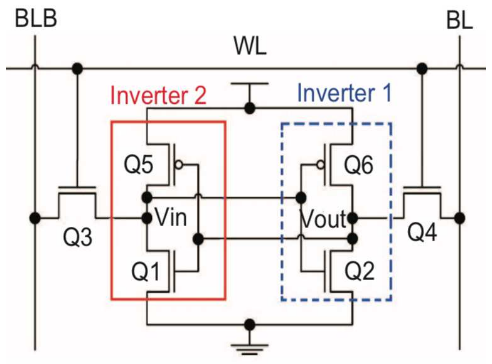


Figure 2.3: A 6-T CMOS SRAM cell [37].

During manufacturing, there are small differences between each SRAM cell due to process variation which leads to a mismatch in the cell [45]. This mismatch also means that the two inverters will always behave distinctly. The mismatch itself does not disturb the normal storage functionality of SRAM cell. Based on this bias, SRAM cells can be classified into three categories as shown below [45]:

1. Non-skewed cell
   A non-skewed cell has no preference during its startup due to the im-

pact of process variations does not cause any mismatch between the two inverters. This cell has a heavily fluctuated start-up value depending upon the noise introduced in the system.

2. Partially-skewed cell
   A partially-skewed cell has a small mismatch between the inverters which lead to a preference over value '0' or '1' but the cell can flip its value upon variation in external parameters.

3. Fully-skewed cell
   A fully-skewed cell is a heavily mismatched SRAM cell in a way that the cell inclined towards value '1' or '0' and has a resistance against external influence/noises.

In ideal SRAM PUF scenario, the utilized SRAM cells should be fully-skewed. Fully-skewed cells lead to a guarantee that the PUF response of a given challenge will have small or no difference even though noises present.

2.6.3 Problem: Noise

Similar to most electronic components, SRAM PUF is also affected by any external influence/noises. These noises will flip unstable bits inside the SRAM PUF. Below are some factors presenting noises:

- Voltage
  The noise introduced by voltage is called power supply noise [47]. This noise is related to changes in the delay characteristics of the gate. The changes will occur when there are switchings in the circuit after the device is turned on which increase dynamic power and cause a voltage drop on power lines and voltage increase on ground lines.

- Temperature
  Temperature variation can be introduced by the surroundings or voltage variation. The preference of a cell inside SRAM has a high probability to be affected by temperature [45].

- Crosstalk
  Crosstalk appears when a signal transmitted on a circuit introduces unwanted side effects in another circuit. Crosstalk happens due to a tight gap between the SRAM cell (tiny interconnect spacing and width). This event becomes more popular due to wider use of faster-operating speeds and smaller geometries (advancement in nanometer technologies) which lead to higher density. Crosstalk is a major contributor to signal integrity problems in modern designs [47]. In addition, higher density in SRAM also influences how environments

affect SRAM performance (more prone to voltage and temperature difference) [48].

- Aging
  Aging is related to changes in the silicon after usage for a long time [49]. There are three main effects related to the aging of a circuit; time-dependent dielectric breakdown (TDDB), bias temperature instability (BTI) and hot carrier injection (HCI) [50]. TDDB is associated with the creation of a conduction path through the gate transistor structure which causes an increase in power consumption and the circuit delay [51]. BTI causes a degradation of the transistor threshold voltage [52]. HCI generates a change in the transistor threshold voltage [53]. HCI is caused by a high current in the transistor channel injecting charges into the gate oxide during the switching.

2.6.4 Bit Selection Algorithm

As mentioned before, during enrollment, challenge-response pairs are gathered. In SRAM PUF, there are two types of challenges that can be applied to the system. The challenge can be either the whole SRAM memory or specific addresses. If a set of addresses is given as a challenge, an address in there can refer to an address of a byte, a bit, or a sequence of bytes or bits.

If specific addresses of SRAM cells are used for PUF challenge, one of the major steps on using SRAM PUF is looking for stable bits. Stable bits itself refers to fully skewed cells explained before. Even though the error correction code is present to correct the noise of bit responses, it also has a limitation on how many bits it can correct. Choosing the most stable bits is important to ensure that the PUF result is always the same throughout its lifetime. Below we present two known algorithms to search for stable bits:

1. Neighbor Analysis
   The first algorithm is using the rank of total stable neighbors which proposed by Xiao et. al. [54]. They argue that the cells which are most stable across environmental conditions are surrounded by more stable cells during enrollment. A stable cell surrounded by more stable cells has a tendency to become more stable because its neighboring cells are likely to experience similar aging stress and operating conditions. In this algorithm, all the stable cells are given weight according to the number of stable bits surrounding it. The more stable neighbor cells it has, the higher weight it gets. For example, if a cell is not stable, it is given zero as its score. If it is stable, at least it will get score one. If it only has one stable neighbor on each left and right side, it will get score two as result of an addition of one from being a stable cell and one from having a stable neighbor on both sides. To get score three, it

needs to be stable and has two stable neighbors on left and right sides. After determining the weight of each cell, a heuristic algorithm that greedily chooses cells for the PUF ID/key with weight greater than a threshold is used.

Before the algorithm is performed, one should collect lots of SRAM cells value first. The data should be retrieved in various condition, for example, different voltages, temperatures, and time differences between enrollment. Afterwards, using the data gathered, the location of all stable bits in SRAM need to be located. A stable bit has to has the same value in all enrollment. Last, the neighbor analysis algorithm is performed to get the most stable bits in SRAM.

2. Data Remanence Approach
   Another bit selection algorithm is by using data remanence of SRAM cell [55]. There are only two remanence tests involved in this approach: first, writing a value (1 or 0) to the whole memory and second, briefly turning off the power until a few cells flip. The most robust cells are the cells which effortlessly flipped when written with the opposite data. Strong 1's are bits that are flipped fast after 0 is written to its location. On the contrary, if 1 is written to a bit location and the bit flipped fast, it means that the bit is a strong 0. When using this approach, one should carefully determine the temporal power down time. On one hand, if the temporal power down period is too little, then the data will stay in the previously written state. On the other hand, if the temporal power down time is too lengthy, then the data written in the array will disappear and the SRAM values will go back to its uninitialized state.

   A significant advantage using this algorithm compared to the previous one is a much shorter time required to locate stable bits. Using neighbor analysis, there are many SRAM values need to be gathered first which might take hours or days. Locating stable bits from hundreds of data probably also take time as well. If data remanence approach is utilized, there is no need to gather many data. One only need to determine the temporal power down required to get strong bits required. Since usually the temporal down period required is less than 0.5 seconds, this analysis only takes few minutes.

2.7 PUF Applications

In this section, we present three applications which are constructed based on PUF technology. The first application is about generating a key using SRAM PUF, the second one is related to secret key binding based on fuzzy commitment scheme, and the last application is secure key storage using optical PUF and coating PUF.

2.7.1 Key Generation using SRAM PUF

In this section, there are two schemes for key generation presented. Both constructions were built by Hyunho Kang et. al. in 2014. The first construction, shown in Figure 2.4, utilizes random number generator (RNG). In this example, a key is produced by applying SHA256 hash function on a result of XOR operation between PUF response and a random number. The helper data is generated by XOR-ing PUF response with an encoding of a randomly generated number. This design was perfected in the second design shown in Figure 2.5. In the second design, random number generator was removed to make the construction more efficient without affecting the security. In this design, a key is directly generated based on the PUF response while the helper data is created by XOR-ing the encoding result of PUF-generated key with the PUF response. Both designs use BCH codes as the error correcting codes with block length (n) of 255.

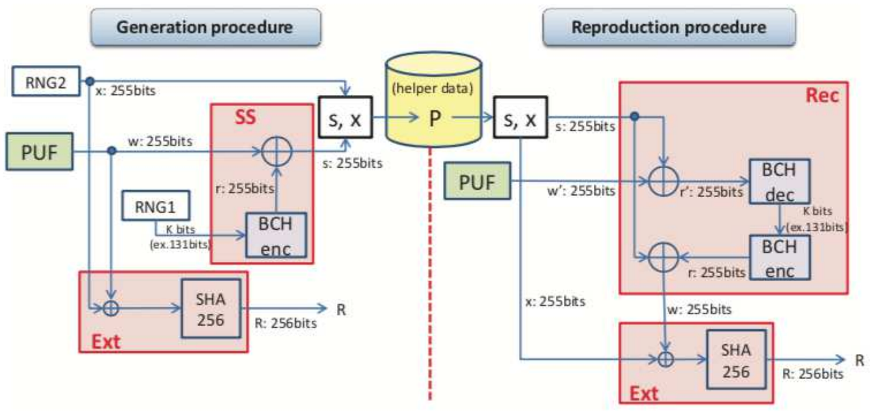


Figure 2.4: Implementation diagram using fuzzy extractor (N = 255) [56].

2.7.2 Secret Key Binding based on Fuzzy Commitment Scheme

Fuzzy commitment was originally introduced by Juels and Wattenberg in 1999 [58]. An example of fuzzy commitment application in PUF domain is presented in [59]. Figure 2.6 shows the flow of this scheme. To securely bind the secret, the secret key S^K needs to be chosen first. Afterwards, the secret key is encoded into a binary codeword C^N. Then, the helper data M^N is generated by masking (XOR-ing) the codeword with the PUF value

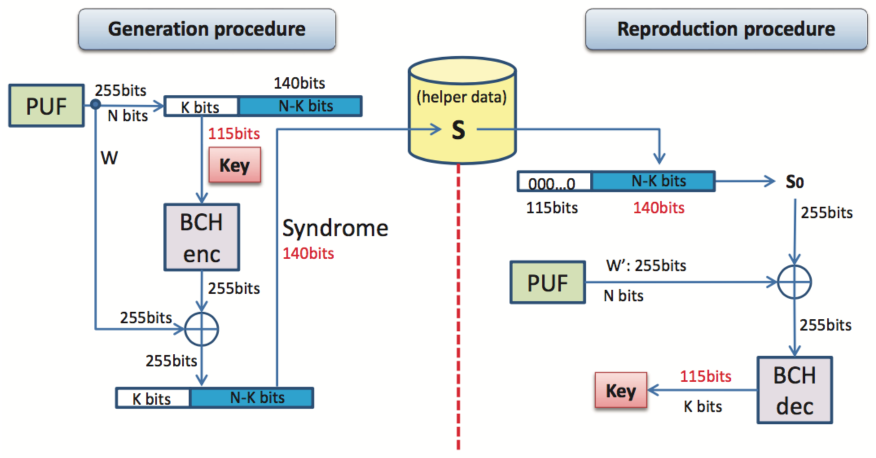


Figure 2.5: Implementation diagram for efficient fuzzy extractor based on the syndrome (N = 255) [57].

X^N. To reconstruct the secret, a noisy version of the codeword $\widetilde{C}^N$ need be calculated by masking the helper data with the noisy version of PUF observation Y^N. The secret $\widehat{S}^K$ can be regenerated by decoding the $\widetilde{C}^N$.

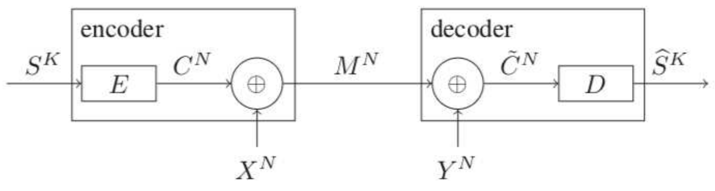


Figure 2.6: Fuzzy commitment scheme [59].

2.7.3 Secure Key Storage using Optical PUF and Coating PUF

In [60], Skoric et. al. present a secure key storage scheme using two extrinsic PUFs; coating PUF and optical PUF. Coating PUF technology is built upon on-chip capacitive quantifications of arbitrary dielectric characteristics of a covering layer which located on top of an IC [61]. Optical PUF itself consists of a 3-D physical structure containing randomly distributed light-scattering particles that produces a speckle pattern (response) when irradiated with a laser beam [60]. This speckle pattern can be considered as the unique fingerprint of the structure. Both PUFs are also considered a strong PUF (has a large CRPs), but optical PUF is considered to be superior than coating PUF due to a much higher number of CRPs and more entropy per response.

In their scheme, to securely store the key, they proposed to store the long-term key in encrypted form. To access the long-term key, a short-term key extracted from the PUF is required.

2.8 Previous Experiments on Off-The-Shelf SRAM PUF

There are many experiments related to SRAM PUF which are performed on off-the-shelf SRAM. Most of these experiments are using off-the-shelf SRAM that are embedded in a microcontroller. For example, in [62], Herrewege et. al. demonstrate a testing of SRAM characteristics on five different microcontrollers; ARM Cortex-A, ARM Cortex-M, Atmel AVR, Microchip PIC16 and Texas Instruments MSP430. They show that not every SRAM embedded in a microcontroller is ideal for an SRAM PUF such as Microchip PIC16F1825. Fortunately, the other microcontroller's SRAMs show an acceptable result to be a PUF candidate (is stable, unique and has enough randomness). Another example is a work done by Anagnostopoulos et. al. [63] in which they present low-temperature data remanence attacks against intrinsic SRAM PUFs, specifically ARM Cortex-M4F LM4F120H5QR microcontroller.

Even though not as many as experiments done on microcontroller's SRAM, there are also some related works that doing the experiments using off-the-shelf SRAM that is not embedded in a specific device. Akhundov in [64] presents a concept of using SRAM Microchip 23LC1024 as the root-of-trust of his public-key based authentication architecture. He shows the result of HD_{intra}, HD_{inter} and the distribution of 0's and 1's experiment of Microchip 23LC1024. Unfortunately, the testing was not performed in various condition (different voltage, temperature, and aging effect). Schrijen and van der Leest in [65] shows a comparative analysis of seven different SRAMs which manufactured using different technology; Cypress CY7C15632KV1 8 (65nm), Virage HP ASAP SP ULP 32-bit (90nm), Virage HP ASAP SP ULP 64-bit (90nm), Faraday SHGD130-1760X8X1BM1 (130nm), Virage asdsrsnfs1p1750x8cm16sw0 (130nm), Cypress CY7C1041CV33-20ZSX (150nm), and IDT 71V416S15PHI (180nm). All of them are tested on the reliability (temperature and voltage variance) and uniqueness (HD_{inter} and hamming weight). The results between each SRAM type is different but it can be summarized that all of the tested SRAM memories are suitable as a PUF candidate. Another interesting result from these work is the fact that the most reliable SRAM is achieved by IDT 71V416S15PHI followed by Cypress CY7C1041CV33-20ZSX and Cypress CY7C15632KV18. Another publication is presented by Holcomb et al. [66] where they show start-up measurements from ISSI SRAM, TI microcontrollers, and Intel WISP devices. Unfortunately, the manufacturing technology on these devices is not mentioned.

2.9 Conclusion

It is a challenging task to design a secure data and key storage based on SRAM PUF technology, especially since to solve this task, we need to understand the term of software and hardware security. Therefore, we investigate several background theory related to security, cryptography, and PUF. We started by presenting four elements on common security; confidentiality, authentication, integrity, and non-repudiation. Then, the history of cryptography is explained, followed by explanations on symmetric cryptography, key derivation function, and multi-factor authentication. Later on, theories related to PUF, hamming distance, error correcting codes and SRAM PUF are described. We continue the chapter by a portrayal of several PUF applications, such as key generation and key storage. We also show some previous works related to SRAM PUF which are built using off-the-shelf SRAM. In the next chapter, our proposed ideas on how to build a secure data and key storage using SRAM PUF and off-the-shelf SRAM will be explained.

Chapter 3

Proposed System

This chapter contains our proposed system to achieved our goals which explained in Chapter 1. First, assumptions, requirements and steps to achieve thesis' main goal on our system are presented. The chapter continues with reasonings on our chosen embedded platform, Arduino. Then, the selected error correcting code which will be used in the system is explained. Afterwards, we present our data and key storage scheme and also our way to generate key using SRAM PUF. Last, our idea to use bits locations as a PUF challenge is shown.

3.1 Assumptions, Requirements and Steps To Achieve Main Goal

To focus the thesis approach, we have defined several assumptions. First, the field of the constructed SRAM PUF application is decided to be only available offline. Accessing the SRAM PUF requires the user to have the device next to his/her side. Second, an attacker cannot access the SRAM directly. An attacker may gain the knowledge of the helper data and challenge used in the PUF concept. Last, there is no analysis and/or solution against physical attacks, e.g side channel attacks, in our secure data and key storage scheme. The scheme is designed to be secure against theoretical attacks.

We also have define a set of requirements related to our system. Below are the requirements defined:

1. Software-based construction
   There should be no major hardware modification or hardware design to implement the project.

2. Patent/license free
   Any dependent component of the design should be in public domain.

3. Open-source and collaboration oriented
If there is a reliable open source project which can be a foundation for this thesis project, instead of building our own software, it is preferred to use that project. This will significantly reduce the time consumed on constructing the whole project. Using other project source code can also increase the collaboration atmosphere. In addition, this requirement may help this project to be known by others since they might introduce our project as one of the projects that uses their code.

4. Key-length security level
The goal on the key-length security level is 256-bits. The concept constructed should be able to use this level and the project's security should be uncompromised even though the key-length is only 256-bits.

5. Off-the-shelf SRAM
The SRAM involved in the thesis should be easily available in the market and cost insignificant.

6. Affordable
The total hardware required to produce the system should be inexpensive.

7. Reproducible
Anyone should be able to reproduce this thesis experiment with no significant effort.

Another thing that we would like to address is steps required to achieve this thesis' main goal which related to answering this thesis' problem statement. As shown in Section 1.3, the problem statement is "How to develop an open-source secure data and key storage scheme using off-the-shelf SRAM component and software-based SRAM PUF technology?". Below are several steps expected to be done as an attempt to achieve this goal:

1. Choose embedded platform on where the system will be built.

2. Select a type of error-correcting codes which will be used as an element in the key generation scheme. The memory required by the error correcting codes has to fit in the embedded platform's internal memory.

3. Search and analyze existing SRAM PUF-based key generation schemes. Propose one of them to be used as the key generation procedure in this thesis and also calculate the maximum error rate that can be handled by the key generation scheme.

4. Propose and construct a system to enable secure data and key storage based on the selected key generation scheme and SRAM PUF technology. Any library required in the system has to be open-source. In addition, the constructed system has to work without any explicit hardware design, in other words, *software-based* construction.

5. Get off-the-shelf SRAM components available in the market.

6. Locate the stable bits inside each SRAM using bit selection algorithm.

7. Test the reliability of SRAM's stable bits. The error rate of the stable bits (referred as HD_{intra}) has to be lower than the maximum error rate that can be handled by the key generation scheme.

8. If the SRAM's stable bits is reliable, check also the uniqueness of the SRAM by ensuring that there is no overlap between HD_{intra} and HD_{inter}.

9. Continue using this SRAM as the root-of-trust in the proposed secure data and key storage if it is proven to be reliable and unique.

10. Test the complete secure data and key storage scheme in various scenarios.

3.2 Arduino Mega 2560 as the Embedded Platform

One of the important details of our system is choosing the platform on where the system will be built. There are two major candidates, Arduino and Raspberry Pi. Both are chosen due to its popularity, availability (easy to get), and various types available. High popularity means the debugging process can be done fast and many references are available online to help the system development. Availability is important because one of the thesis' goals is to be easily used by anyone. Low availability will reduce the reusability of this project and user's interest. Various types available is a pleasant option for system flexibility. For example, if a user wants to develop a more complex system on top of this thesis' system or desire to use a more complex error correcting codes, he/she can choose a platform with higher computing capability. Besides those three factors, another feature which lead on selecting Raspberry Pi and Arduino is their GPIO. GPIO availability will enable easy communication between the SRAM and the embedded platform.

Compared to Arduino, Raspberry Pi offers a higher computing capability and relatively easier development. This is because Raspberry Pi is basically a mini Linux computer. One can develop a software using C, C++, Python, etc. in Raspberry Pi which may fasten the project development, especially for a developer who already familiar with a specific programming language. Unfortunately, Raspberry Pi requires a longer startup time compared to Arduino. It also requires higher electrical power. If one wants to use the developed project in the embedded area, this two factor is a major trade-off.

Due to the these considerations, Arduino is chosen. Even though one has to construct the system in C++, this can be a positive thing since one can maximize the computing capability easily. Moreover, Arduino itself is an open-source project which enable anyone to develop their own boards and software libraries [67].

There are various Arduino types available on the market. The chosen Arduino type is Arduino Mega 2560. It is selected because it offers larger memory capability compared to other types, such as 256k bytes of Flash memory, 8k bytes internal SRAM, and 4k byte EEPROM. Besides, it also has 54 digital I/O pins and 16 analog I/O pins which ease the communication to external SRAM.

3.3 BCH Codes as Error Correcting Codes

As shown in previous chapter, there are two major categories in ECC; soft-decision and hard-decision code. We choose to use hard-decision ECC over the soft decision one because in hard-decision ECC, there is no requirement to provide the error probability function on SRAM cells. Calculating the error probability on SRAM cells will require extra step, overcomplicate the system and the procedure on using the constructed system. Futhermore, between two examples of hard-decision ECC, we particularly pick BCH codes as the ECC due to lower computing resource required compared to Reed-Solomon Codes. BCH also better at correcting random errors than Reed-Solomon Codes.

As mentioned in the previous chapter, BCH codes are flexible Error Correcting Codes (ECC) shown by multiple parameters available. The only fixed parameter is q since the problem is in binary form ($q = 2$). The source code for BCH codes utilized in our construction is a modified version of Robert Morelos-Zaragoza's version which can be retrieved at [68]. This code is selected because it can support m ranging from 2-20 which mean the length of the code that can be corrected ranging from 2 until 1048575. When using BCH codes, one should be careful on deciding the parameters that will be used, for example, larger m or n means a bigger memory needed. These parameters should be determined with several considerations, such as the inner hamming distance of SRAMs and memory available on Arduino Mega 2560.

On deciding the value m, a further look on the memory required during the error correction computation need to be done. Inside the bch codes from [68], the decoding method requires the largest memory compared to other procedures. There are six parameters that depend on m which are elp, d, l, u_lu, s, and err. Table 3.1 shows the required memory given the m value.

Table 3.1: Memory required (bytes) given the value of m.

m	**Bytes Required**	**m**	**Bytes Required**
2	53	12	16805897
3	129	13	67166217
4	377	14	268550153
5	1257	15	1073971209
6	4553	16	4295426057
7	17289	17	17180786697
8	67337	18	68721311753
9	265737	19	274881576969
10	1055753	20	1099518967817
11	4208649		

Since the internal SRAM in Arduino only has 8k bytes capacity, the chosen m is 6 (requires 4553 bytes, around 55% of total SRAM available in Arduino). This parameter will result in possible n between 32 and 63. n is chosen to be 63 to maximize the length code that can be encoded. The combination of m = 6 and n = 63 results in various k and t that can be chosen. The combination of all parameter possible is shown on Table 3.2.

Table 3.2: BCH parameter for m = 6 and n = 63.

k	**t**
57	1
51	2
45	3
39	4
36	5
30	6
24	7
18	10
16	11
10	13
7	15

To maximize the error correction capability, k = 7 and t = 15 is chosen.

All these parameter combination will enable error correction capability 23.8% of the data length. To summarize, here are the chosen parameters:

- n: 63
- k: 7
- d: 31
- t: 15

3.4 Data and Key Storage Scheme

Figure 3.1 shows the scheme to protect user's data and key. On an attempt to protect the user's data and key, our proposal is divided into three major parts, first is generate the final key, and the rest is using the final key either to encrypt or decrypt data. To prevent unauthorized person accessing the data with a stolen PUF, an idea from multi-factor authentication is utilized. Instead of just depending on the PUF device to access the key, a combination of PUF device and user knowledge is presented. User knowledge that used here is password. User's password is combined with the PUF-generated key to generate a **final key** using HMAC. The input message to the HMAC is user's password and the input key to the HMAC is the PUF-generated key. The HMAC function proposed to use is HMAC SHA3 with key length 256 bits. The final key can be used to encrypt and decrypt user data/key. To decrypt and encrypt the data, a symmetric encryption algorithm is preferred over the asymmetric one. The symmetric encryption algorithm used is AES with key length 256 bits and modes of operation CTR with randomly generated IV. To achieve IND-CCA security level, we also implement encrypt-then-MAC technique (explained in Section 2.2.2). The MAC in the encrypt-then-MAC procedure is generated using HMAC SHA3-256 with PUF-generated key as the key.

3.5 Key Generation Scheme

As shown in the previous section, the secure data and key storage scheme requires the PUF to generate the key which will be used to generate the final key. The key generation scheme used in this project is a modified version of Figure 2.5 proposed in [57]. Instead of using n = 255, the scheme used in this project will choose n = 63. The parameter *n, k, t, d* is similar to the parameter chosen in the Section 3.3. Figure 3.2 illustrates the mentioned scheme.

Using this scheme, to generate a key with length 256-bits requires 37 blocks of this scheme, which lead to 2331 bits required. 37 blocks are

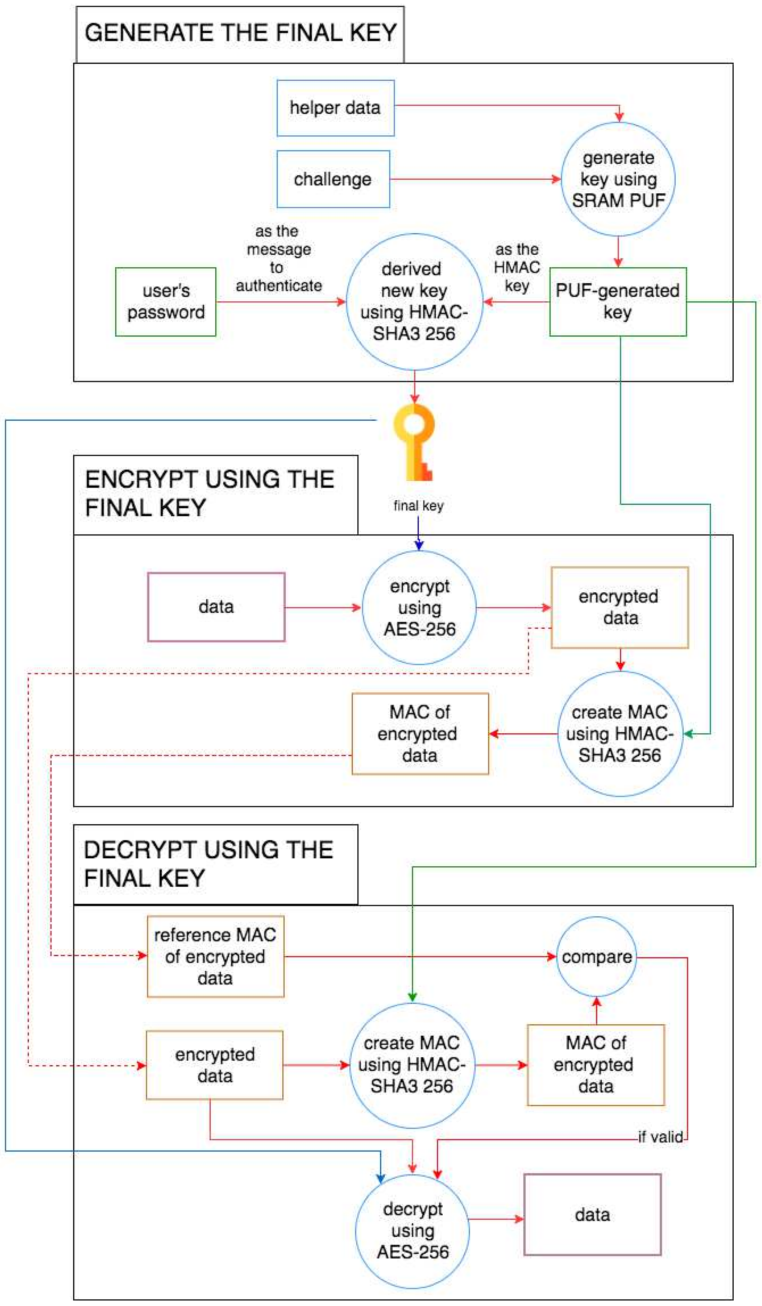

GENERATE THE FINAL KEY
helper data
challenge
generate key using SRAM PUF
as the message to authenticate
user's password
derived new key using HMAC-SHA3 256
as the HMAC key
PUF-generated key
final key
ENCRYPT USING THE FINAL KEY
data
encrypt using AES-256
encrypted data
create MAC using HMAC-SHA3 256
MAC of encrypted data
DECRYPT USING THE FINAL KEY
reference MAC of encrypted data
compare
encrypted data
create MAC using HMAC-SHA3 256
MAC of encrypted data
if valid
decrypt using AES-256
data

Figure 3.1: Scheme for secure data and key storage. There are three stages in here; generate the final key, encrypt using the final key and decrypt also using the final key.

calculated from $\frac{256}{7} = 36.57$, rounded-up resulting in 37. 7 is the length of the key generated from 63 bits of data using this scheme. Since one block needs 63 bits of data, 37 blocks require $37 \times 63 = 2331$ bits. In addition, a further look into the scheme will reveal that there is an entropy loss as many as 7 bits every 63 bits of data input during the generation of helper data. Due to this entropy loss, this scheme can only correct errors on maximum 8 bits instead of 15 bits. Based on this reason, to ensure the key generation scheme always produced the same key, the SRAM component used as root-of-trust has to have maximum error rate (shown by $\mathrm{HD}_{\mathrm{intra}}$) 12.7% (calculated from $\frac{8}{63} \times 100\%$).

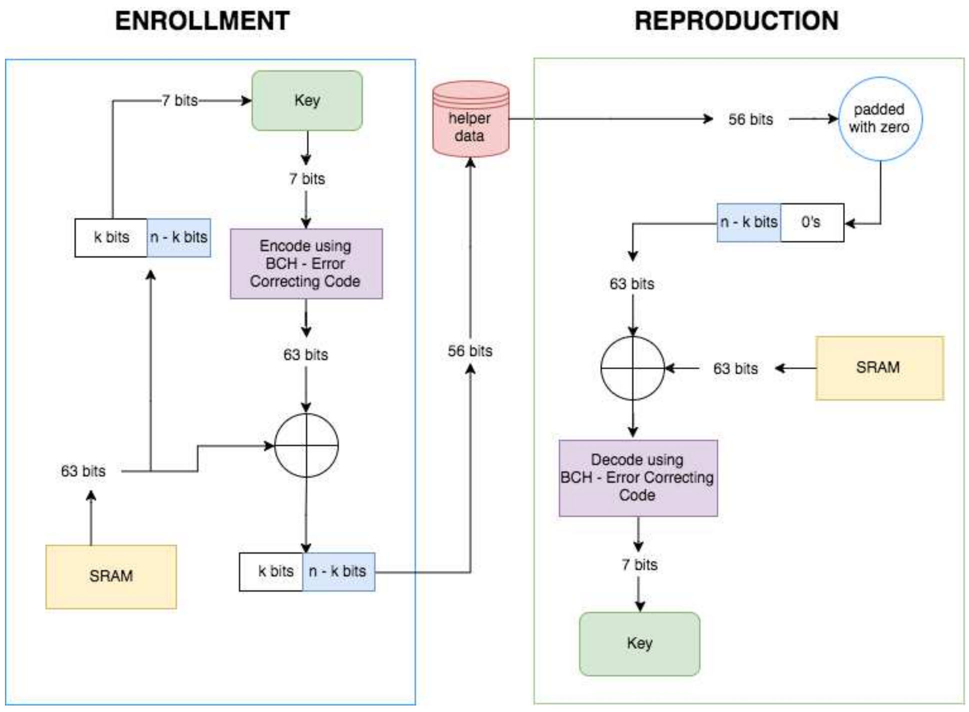


Figure 3.2: Scheme for key generation. $n = 63$, $k = 7$, $t = 15$, $d = 31$.

3.6 Bits Locations as the PUF Challenge - Numerous CRPs

An example of a well known PUF construction which claimed to be resistant against brute force attack is Stanzione and Iannaccone's work [69]. They mentioned that their PUF construction is resistant to 10^{25}-trials brute force attack. Inspired by their work, we imagine having a stronger construction. We envision an SRAM PUF which has total challenge-response pairs possibilities more than the number of atoms on earth which predicted to be at least 10^{49} [70]. To achieve this goal, we come up with an idea to use bits locations as a PUF challenge. Below are the reasons why this decision is taken:

1. Stable bits tend to be scattered all around SRAM memory.
2. If there is a burst error on a bit location inside the challenge, this error will not affect many locations in a challenge since this burst error may only lead to a single location. If a location related to multiple bits is used as the challenge, a burst error will affect many bits generated. For example, if locations of bytes are used as the challenge, a burst error might lead to 8 bits errors in the response generated.
3. There are huge possibilities of challenge-response pairs. The number of possibilities is calculated from the *permutation* of the required bits and the available bits using Equation 3.1.

$$P(n,r) = \frac{n!}{(n-r)!} \tag{3.1}$$

As an illustration, if the number of bits required to generate/reconstruct the key is 2331 bits (the length of the bits required to generate 256-bits key when using the scheme shown in Figure 3.2), then a set of 2331 bits locations is required as an input (a challenge) to PUF device. And if the SRAM has a total capacity of 65536 bits, using Equation 3.1 explained before, there are $P(65536, 2331) = \frac{65536!}{(65536-2331)!} \approx 10^{11209}$ possible combinations. The total possible CRPs is even much higher compared to the total possibilities of the number of bits required ($2^{2331} \approx 5.02 \times 10^{701}$) or the number of possible keys ($2^{256} \approx 1.16 \times 10^{77}$). Due to these large possibilities of challenge-response pairs, this idea will lead to **numerous CRPs**.

Using this concept, before generating the challenge, the location of stable bits needs to be identified first. The location of stable bits can be detected by using bit selection algorithm mentioned in Section 2.6.4. After the location of stable bits is identified, during the generation of a challenge, the locations' order inside the challenge will be randomized.

3.7 Security Analysis of The Proposed Scheme

As mentioned in Section 3.1, our scheme is designed to be secure against theoretical attacks. There are three elements in our scheme as the main parts on ensuring the scheme's security against such attacks; encryption using AES-256, key derivation function using HMAC SHA3-256, and the PUF-generated key. Based on these components, the attack scenarios are presented below:

- The simplest way of attacking the scheme is by applying a cryptanalysis directly to the ciphertext produced by the scheme. If successful, this attempt will result in known final key used in the scheme and the plaintext. In this way, the attacker has to break the security level of AES-256. If brute-force attack is applied to AES-256, the attacker has to try all possibilities of 2^{256} keys which roughly equals to $1.157920892373163 \times 10^{77}$. Even if one can try ten thousand keys every second, the total time needed to try all combinations is still 3.67×10^{65} years (longer than the age of the universe which is 14×10^{9} years old). The attacker may also apply key-recovery attack using a technique called biclique attack [71]. Even though this technique is the best-known attack on AES-256, this technique still requires time complexity of $2^{254.27}$ and data complexity of 2^{40}. In addition, besides cracking AES-256, an attacker has to beat the security of HMAC SHA3-256 as the MAC generator as well. Using this combination (called *encrypt-then-MAC*), it is proven that the security level is IND-CCA [27].

- Another attempt that may be taken by the attacker is by stealing the PUF device. Even though the attacker has the PUF device, he still cannot access the encrypted data directly since he has to guess the PUF owner's password. If PUF owner's password entropy is high enough, the chance for attacker to successfully gain the access to the encrypted data is small since he basically has to break the security level of HMAC SHA3-256 (which requires at least $min(2^k, 2^n)$ time complexity, where k is the key size and n is the hash output size [72]).

- The attacker may also try accessing the encrypted data by doing a social engineering to gain the information of PUF owner's password, then guess the PUF-generated key to get the final key which used to encrypt the data. To successfully predict the PUF generated key, it is also a hard work. For example, if one want to brute force all possible input combinations to the PUF key generation scheme, there are $2^{2331} \approx 5.02 \times 10^{701}$ possible combinations. It is actually easier to just try all possible combinations of the PUF-generated key which has 256-bits in length. Even though such fact exists, the number is not small either. The total possibilities still accounts for 2^{256} keys which roughly equals to $1.157920892373163 \times 10^{77}$ possibilities.

Based on these reasonings, we believe our proposed data and key storage scheme is secure. The only possible way for an attacker to gain information from the encrypted data is by having both PUF device and PUF owner's password.

3.8 Conclusion

After presenting related works in previous chapter, this chapter continues with explanations of our proposed ideas to answer the thesis' problem statement and achieve our goals which explained in Chapter 1. In the beginning, use cases, assumptions, requirements and steps to achieve the thesis' main goal on our system are presented. The chapter continues with reasonings on our chosen embedded platform, Arduino. Specifically, we choose a product type of Arduino called Arduino Mega 2560 because it offers larger memory capability compared to other types and has 54 digital I/O pins which ease the communication to external SRAM. Then, the selected error correcting code which will be used in the system is explained. We choose to use hard-decision ECC over the soft decision since there is no requirement to provide the error probability function on SRAM cells. Between two examples of hard-decision ECC, we particularly pick BCH codes as the ECC due to lower computing resource required compared to Reed-Solomon Codes. BCH also better at correcting random errors. The parameter used in our BCH codes are n: 63, k: 7, d: 31, t: 15. Afterwards, we present our secure data and key storage scheme. To secure the data, we encrypt the data with a final key which derived using HMAC SHA-3 256 as a KDF with input PUF-generated key and user password. To produce the PUF-generated key, we choose to use a modified version of key generation scheme proposed in [57]. We continue by presenting our idea to use bits locations as a PUF challenge. Our proposed PUF challenge are based on the permutation of stable bits which lead to a numerous number of possibilities. Last, we present security analysis of the proposed secure data and key application. There are three elements as the main parts of ensuring the security goal; encryption using AES-256, key derivation function using HMAC SHA3-256, and the PUF-generated key. Based on three elements, we displayed three different attack scenarios. In the next chapter, the experiments done in this thesis will be explained.

Chapter 4

Implementation, Experiments and Results

After describing our proposed system as an attempt to achieve this thesis' goals in the previous chapter, this chapter continues with an explanation of several experiment setups and results. This chapter starts by a presentation on two chosen SRAMs that used in experiments; Microchip 23LC1024 and Cypress CY62256NLL. Afterwards, the testing results on two bit selection algorithms (neighbor analysis and data remanence approach) and the stable bits produced by these algorithms are displayed. The chapter continues with examination on our proposed PUF challenge and a presentation on our complete enrollment scheme. Then, we present the procedure to develop SRAM PUF-based applications using any off-the-shelf SRAM. Next, testing on the designed secure data and key storage scheme is shown. Experiment outcomes on storing Bitcoin private key will conclude this chapter.

4.1 Chosen Off-The-Shelf SRAMs

The first step to do in this thesis implementation is looking for off-the-shelf SRAM components to be the root-of-trust in our SRAM PUF project. Due to numerous SRAM types available in the market, we need to define several requirements for the SRAM first. The main requirements on the SRAM are easy to get (a simple Google search should show some e-commerce websites to buy from), can be bought in small quantity ($\leq$ 5 pieces), stand-alone component (available without buying extra component, e.g. not embedded in an FPGA), inexpensive (cost less than €5), reasonable memory size ($\geq$ 64kb). These criteria are chosen due to some products only sold to a company or an entity that willing to buy in a big quantity or has to be custom made. There are two SRAM types purchased and tested here; Microchip 23LC1024 and Cypress CY62256NLL.

Microchip 23LC1024

The Microchip Technology Inc. 23A1024/23LC1024 is a 1024 kbit Serial SRAM device. This SRAM is very popular, shown by many references available online and several GitHub repositories intended just to access this SRAM. The reason of its popularity can be traced to its cheap price, small size, and easy-to-use. The price is ranging from €1.5-3.5. This device has eight pins which contribute significantly to its small footprint (it has dimension of 9.271 x 6.35 x 3.302 mm). It is easy to use because it provides SPI connection which simplified the communication, and has three modes available; SPI (Serial Peripheral Interface), SDI (Serial Dual Interface) and SQI (Serial Quad Interface). Its voltage range also quite large, ranging from 2.5-5.5V. Figure 4.1 shows the Microchip 23LC1024.

Figure 4.1: SRAM Microchip 23LC1024 [73].

Cypress CY62256NLL

The Cypress CY62256NLL is a 256 kbit SRAM device. Even though this device is less popular than Microchip 23LC1024, it is still widely used. One of the reason is that this device has an automatic power-down feature, reducing the power consumption by 99.9 percent when deselected. Unlike Microchip 23LC1024, Cypress CY62256NLL does not have an SPI connection which complicates the communication. To communicate, one should utilize its twenty-eight pins available. Since it has many pins, this contributes to its significantly larger size compared to Microchip 23LC1024. Specifically, its size is 37.592 13.97 4.953 mm and produced using 90nm technology. Its voltage range is ranging from 4.5V-5.5V. Figure 4.2 shows Cypress CY62256NLL.

4.2 Automated PUF Profiling System

To increase the experiment's efficiency, an automated PUF profiling system is constructed. The system consists of a PC, act as a master, and an Arduino connected to an external SRAM component which acts as a slave. A custom protocol was designed to communicate between them. It is specifically

Figure 4.2: SRAM Cypress CY62256NLL [74].

designed to be generic and usable for all types of PUF profiling measurements. The software on Arduino side waits for measurement commands sent by PC on the serial link after booting. The designed protocol is dedicated for read bytes, write bytes, and SRAM disable/enable. The system also supported parallel profiling which significantly increases the effectivity. Figure 4.3 shows the setup and the schematic to profile four SRAMs Cypress CY62256NLL concurrently using four Arduino.

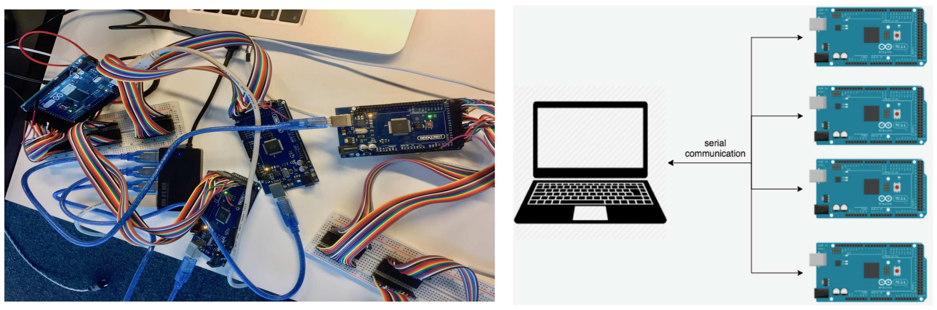


Figure 4.3: Automated PUF profiling setup using a PC and four Arduino. Left picture shows the actual setup, while the right picture displays the schematic of such setup.

4.3 Testing on Chosen Off-The-Shelf SRAMs

As mentioned in Chapter 2, to be qualified as a PUF candidate, an SRAM has to be stable in various conditions. This means if it is given various power input or used in varied temperatures or utilized for a long time, the initialized SRAM values has to remain similar or only has little changes (HD_{intra} has to be lower than the maximum error correction capability of the key generation scheme mentioned in Section 3.5, 12.7%). Under any condition, there should be no overlap between HD_{intra} and HD_{inter}. Moreover, the SRAM has to have a sufficient amount of randomness, shown by having equal distributions between 1's and 0's on its values. To ensure the quality of these two SRAMs, there are several experiments performed on each SRAM, such as calculating HD_{intra} and HD_{inter} given the whole memory value as the challenge and also the distribution of 1's and 0's inside SRAM memory.

Microchip 23LC1024

There are ten SRAMs Microchip 23LC1024 that were available during the experiment. To check whether this SRAM is a justifiable candidate for PUF, several testings are performed. First, the number of 1's and 0's in memory after a start is calculated. Unfortunately, the average distribution of 1's and 0's are not similar, 1's occupy 70% and 0's fill the remaining 30%. Second, HD_{intra} and HD_{inter} are calculated on these chips. The calculation is done using twenty memory values on each chip which retrieved at room temperature, 5V input and 10 seconds interval between retrieval attempts. From these chips, the average HD_{intra} is 6.18% and the average HD_{inter} is 42.54%. Third, the effect of voltage variation on the HD_{intra} and HD_{inter} are also evaluated. The calculation is done using memory values on each chip which retrieved on room temperature and 10 seconds interval between retrieval attempts. The voltage range is between 2.5V and 5V with 0.5V increase on a step. On each step, there are three data retrieved. Using these data, voltage variation results in an average HD_{intra} 8.21% and an average HD_{inter} 42.59%.

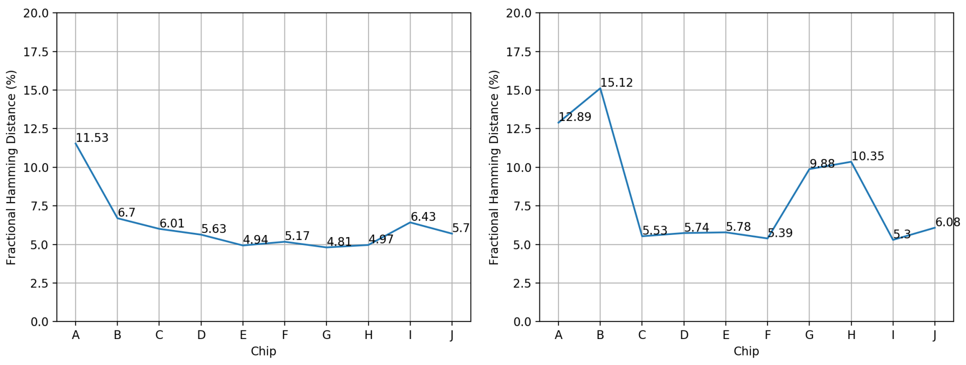


Figure 4.4: HD_{intra} of ten SRAMs Microchip 23LC1024. The left figure is the testing result of HD_{intra} with constant voltage, the right one is tested based on the voltage variation.

Based on these experiments, SRAM Microchip 23LC1024 shows questionable results. First, the distribution of 1's and 0's inside the SRAM is not balanced. Second, a voltage variation shows that it significantly affects the HD_{intra}. Third, there are two SRAMs Microchip 23LC1024 ('A' and 'B') that shows HD_{intra} larger than the maximum error correction capability of the key generation scheme (12.7%) when they are tested on the effect of voltage variation. Fortunately, there is no overlap between HD_{intra} and HD_{inter}. Even though these outcomes make us doubtful on this SRAM quality as an SRAM PUF candidate, we decided to continue using this SRAM in further experiments. Hopefully, when we locate the stable bits inside the SRAM, the experiments done on the stable bits will show a better result than this result.

Cypress CY62256NLL

There are five SRAMs Cypress CY62256NLL that were available during experiment. Similar like on previous SRAM, several testing are performed to check whether this SRAM type is a justifiable candidate for PUF. First, the number of 1's and 0's in an initialization is counted. Fortunately, unlike the 23LC1024, the average distribution of 1's and 0's are similar, both occupy 50% of total bits available. Next, HD_{intra} and HD_{inter} are calculated on both chips. The calculation is done using twenty memory values on each chip which retrieved at room temperature, 5V input and 10 seconds interval between retrieval attempts. From these chips, the average HD_{intra} is 4.85% and the average HD_{inter} is 39.28%. Last, the effect of voltage variation on the HD_{intra} and HD_{inter} are also evaluated. The calculation is done using chip memory values on each chip which retrieved on room temperature and 10 seconds interval between retrieval attempts. The voltage range is between 4.5V and 5V with 0.1V increase on each step. On each step, there are ten data enrolled. The average HD_{inter} on voltage variation is 38.59%, while HD_{intra} is 3.58%. Figure 4.5 shows the HD_{intra} between the constant and the variated voltage.

The results shown before indicate that SRAM Cypress CY62256NLL is a qualified candidate for SRAM PUF. A well distributed 0's and 1's inside SRAM memory, voltage variation has little effect on HD_{intra} and HD_{inter}, and no overlap between HD_{intra} and HD_{inter} lead us to continue using this SRAM for further experiments.

4.4 Testing on Bit Selection Algorithms

In this section, the test on stable bits produced by two algorithms, neighbor stability and data remanence analysis, is shown. The test was done on a single chip of each SRAM type. The explanation of both algorithms can be found on Section 2.6.4.

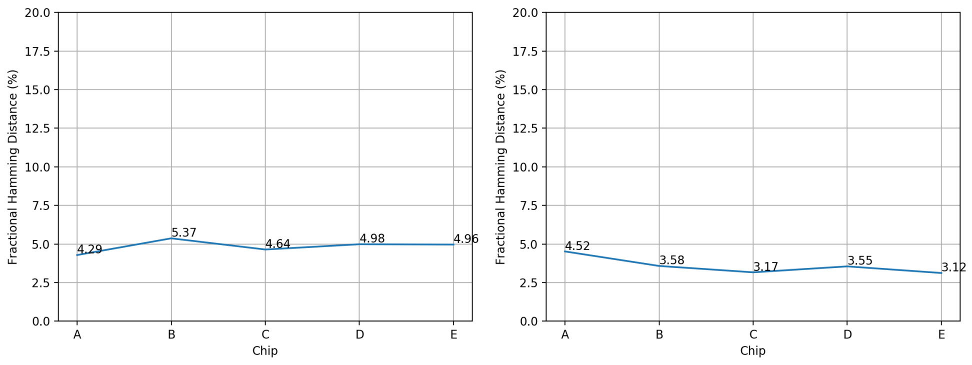


Figure 4.5: HD_{intra} of five SRAMs Cypress CY62256NLL. The left figure is the testing result of HD_{intra} with constant voltage, the right one is tested based on the voltage variation.

4.4.1 Neighbour Stability Analysis

To use this algorithm, first, data of SRAM bits value from various conditions (voltages and time difference between data retrieval attempts) need to be gathered. Afterwards, the bits which remained stable over all retrieved data are located. Then, the rank of remained stable bits are calculated. Last, n bits with highest rank can be used according to the necessity. The higher the rank, the more stable that bit should be.

Microchip 23LC1024

As input for the algorithm, there are 500 data of SRAM bits value used for this chip. The voltage variation is randomized between 2.5V - 5.0V. The time difference between data retrieval attempts is ranging from 5 seconds until 1 hour. SRAM Microchip 23LC1024 itself has capacity 1048576 bits. After doing the calculation from those five hundred data, there are 413374 remaining stable bits. From those remaining stable bits, the rank of each bit is calculated. The frequency of bits rank is shown in Figure 4.6. As shown in this figure, the total bits with rank more than 5 is insignificant, only showing 493 bits. Bits with rank more or equal to six is merged into a single bar because the frequency among those rank is usually only a single digit.

Cypress CY62256NLL

Similar like with SRAM Microchip 23LC1024, there are 500 memory values retrieved in SRAM Cypress CY62256NLL. Cypress CY62256NLL is able to store 262144 bits in its memory. The remained stable bits after

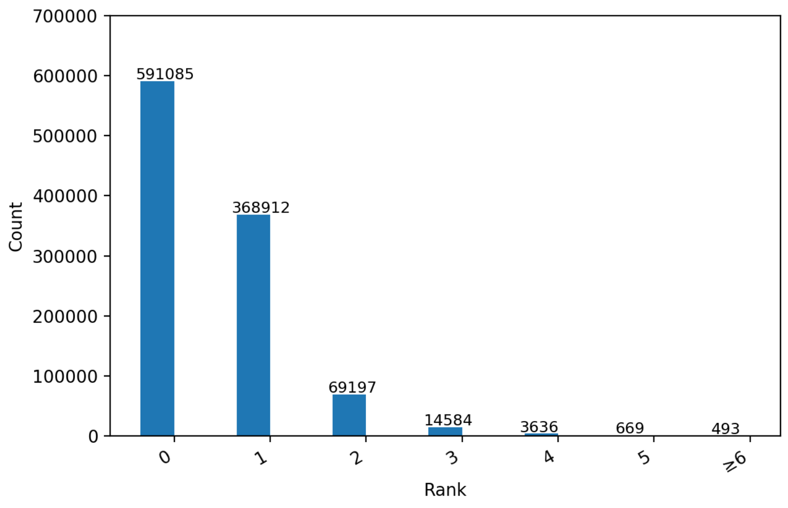


Figure 4.6: Remaining stable bits count according to their rank in SRAM Microchip 23LC1024.

500 data retrieval are 102708 bits (39,18%). The result of the calculation is shown on Figure 4.7. Compared to Microchip 23LC1024, this SRAM shows more promising result since there are many bits with ranks more than seven. Even to get two thousand stable bits, the lowest rank that can be included is twelve.

4.4.2 Data Remanence Approach

The result of data remanence analysis on both SRAMs is shown below.

Microchip 23LC1024

On SRAM Microchip 23LC1024, the data remanence analysis is done on time variance between 0-1.0 second. The result can be seen on Figure 4.8. In this figure, it is shown that SRAM Microchip 23LC1024 will reach the uninitialized point if it is temporarily turn off for 0.7 seconds.

Cypress CY62256NLL

On SRAM Cypress CY62256NLL, the data remanence analysis is done on time variance between 0-10 seconds. The result can be seen on Figure 4.8. In this figure, it is shown that SRAM Cypress CY62256NLL will reach the uninitialized point if it is temporarily shut down for 5.0 second.

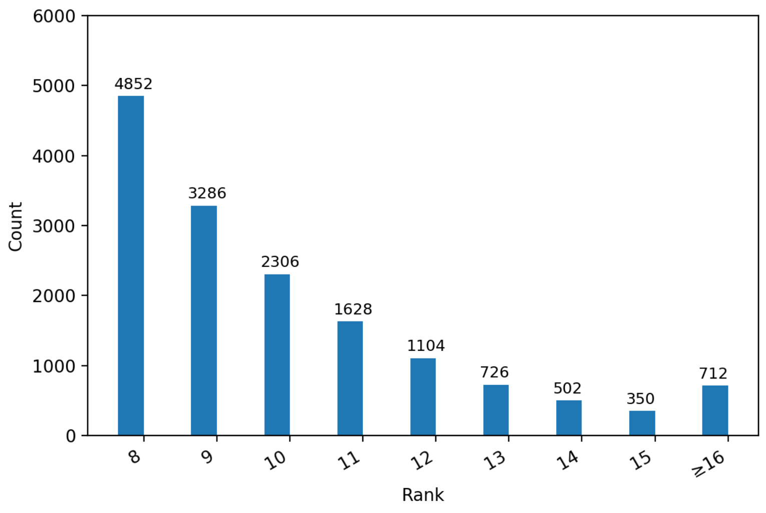


Figure 4.7: Remaining stable bits count according to their rank in SRAM Cypress CY62256NLL. There are 246678 bits with rank less or equal to seven.

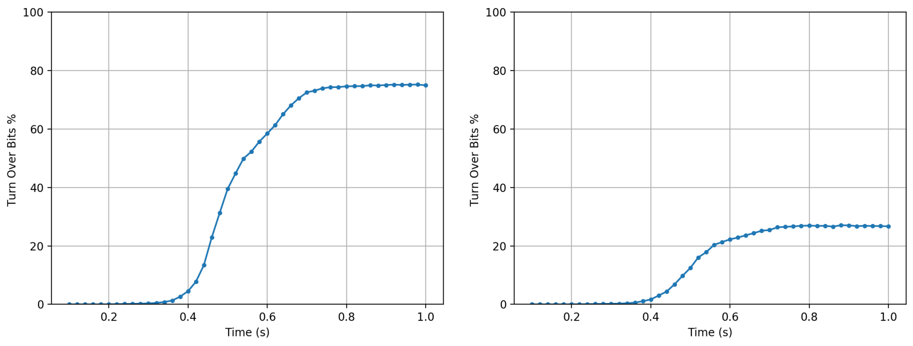


Figure 4.8: Measured SRAM Microchip 23LC1024 data remanence for data 0 (left) and data 1 (right). SRAM Microchip 23LC1024 will reach the uninitialized point if it is temporarily shut down for 0.7 second.

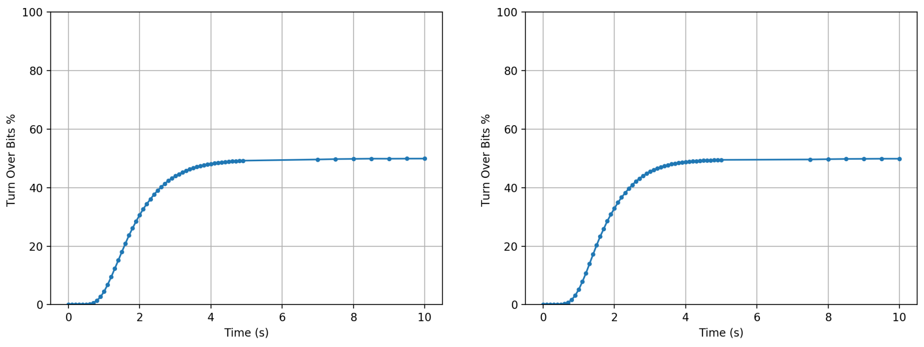


Figure 4.9: Measured SRAM Cypress CY62256NLL data remanence for data 0 (left) and data 1 (right). SRAM Cypress CY62256NLL will reach the uninitialized point if it is temporarily shut down for 5.0 second.

4.4.3 Stability Test on Stable Bits

In this section, test results on the effect of time interval and voltage on stable bits using both algorithms on each SRAM are shown. The effect of aging and temperature is not tested due to a limitation on time and equipment. For the effect of time interval testing, the enrollment was done on 16 days with one day gap between enrollment. Voltage effect testing was done on voltage ranging from 4.5V-5V for SRAM Cypress CY62256NLL and 2.5V-5V for SRAM Microchip 23LC1024. The test is done on 4662 bits which is twice the length of the bits required to generate 256 bits key when using scheme shown in Figure 3.2. The result of time interval testing on SRAM Microchip 23LC1024 is shown on Figure 4.10, while Figure 4.12 displays the result for SRAM Cypress CY62256NLL.

Microchip 23LC1024

- Neighbor Stability Analysis
  To get 4662 bits, there are three categories included; rank similar or higher than 6 with 493 bits, rank 5 with 669 bits, and rank 4 with 3500 bits. During testing on variated voltage and time interval, the stable bits generated using neighbor stability analysis show a poor performance by having maximum 2389 bits changing (HD_{intra} 51.24%) which also produces an overlap between HD_{intra} and HD_{inter}. The maximum difference is produced when the difference between enrollment is 8 days.

- Data Remanence Approach
  To get 4662 bits, strong 1's are generated using power down period of

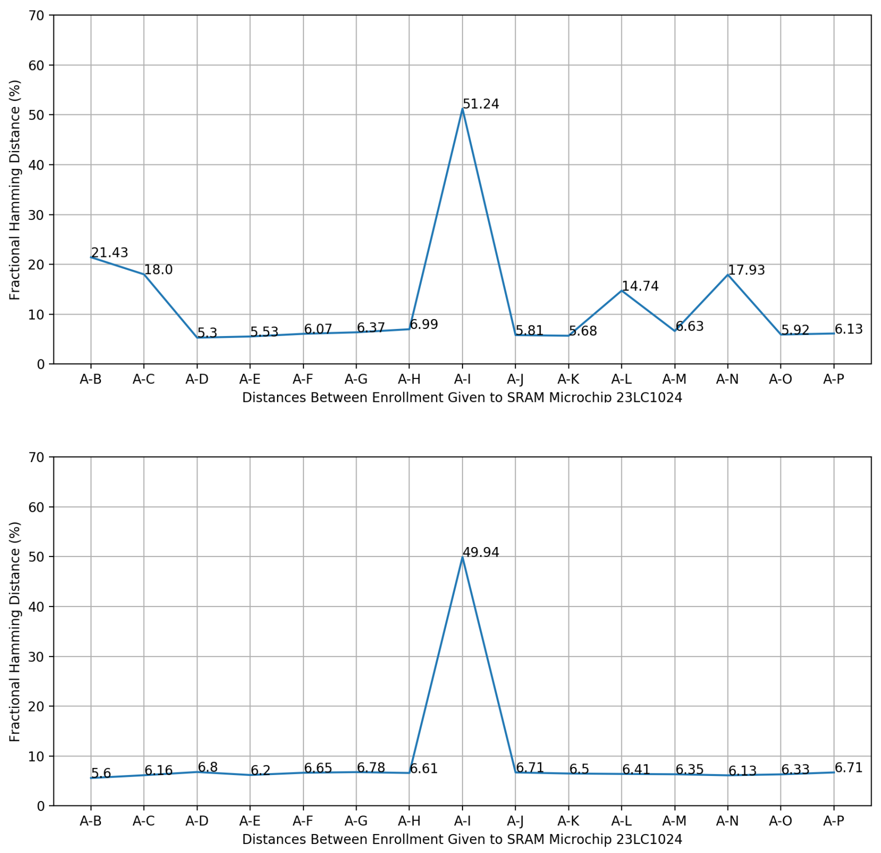
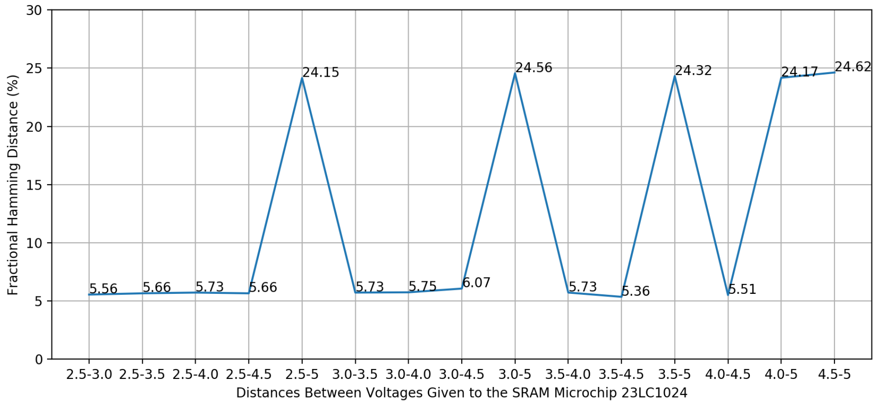


Figure 4.10: Time interval testing results on SRAM Microchip 23LC1024. The top figure is the testing result on stable bits generated using neighbor analysis, while the bottom one is tested on stable bits generated using data remanence approach. Index A on x-axis refers to enrollment on day 1, B on day 2, etc. Index A-B refers to fractional hamming distance between enrollment on day 1 and day 2.



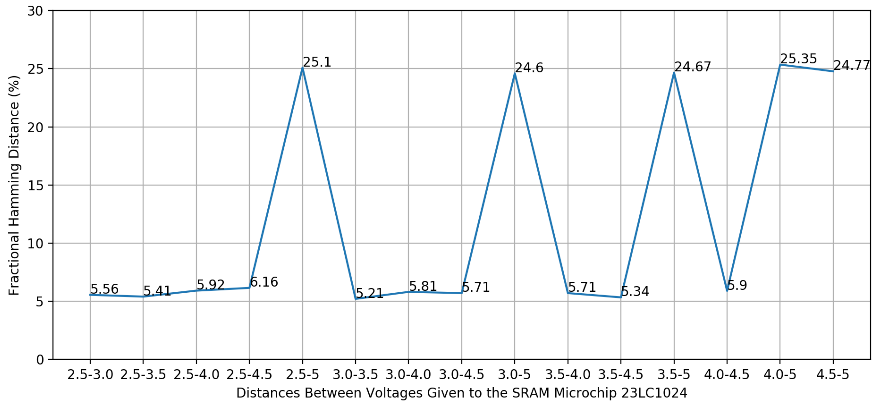
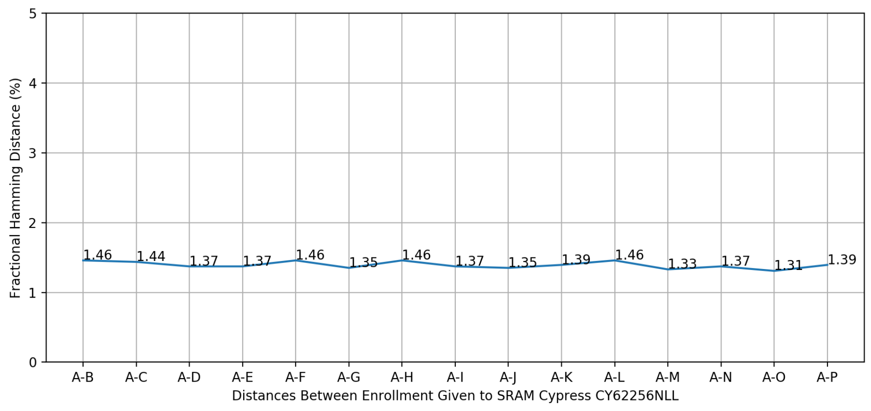


Figure 4.11: Voltage variation testing results on SRAM Microchip 23LC1024. The top figure is the testing result on stable bits generated using neighbor analysis, while the bottom one is tested on stable bits produced by data remanence analysis. Index on x-axis refers to two different voltages, e.g. 2.5-5.0 means the fractional hamming distance between enrollment on voltage 2.5V and voltage 5.0V.



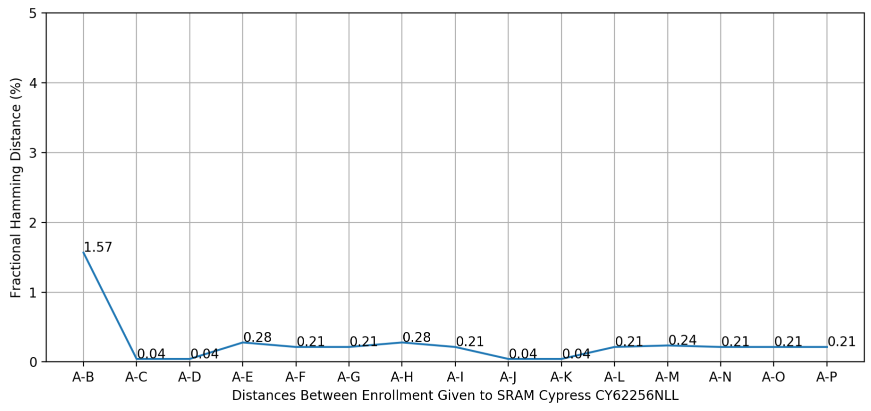
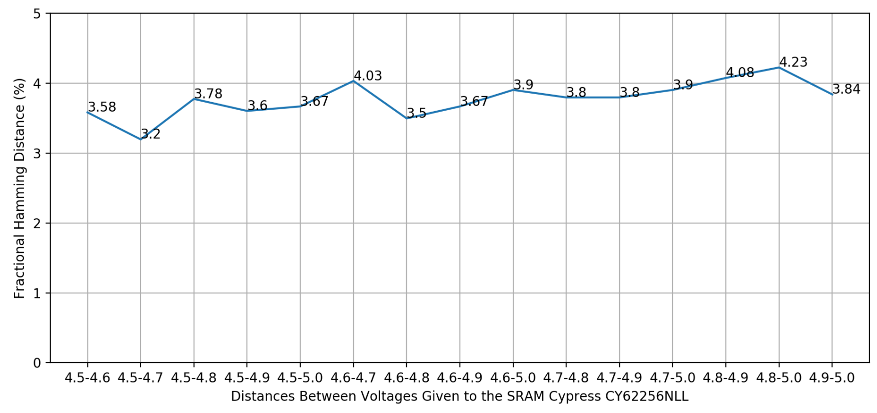


Figure 4.12: Time interval testing results on SRAM Cypress CY62256NLL. The top figure is the testing result on stable bits generated using neighbor analysis, while the bottom one is tested on stable bits generated using data remanence approach. Index A on x-axis refers to enrollment on day 1, B on day 2, etc. Index A-B refers to fractional hamming distance between enrollment on day 1 and day 2.



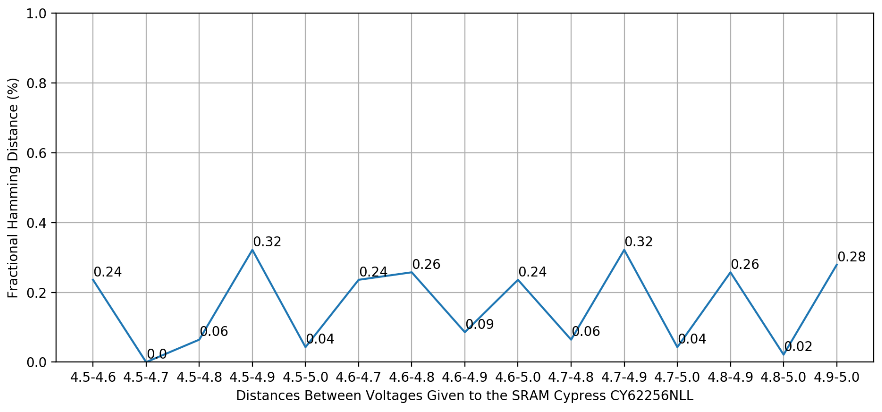


Figure 4.13: Voltage variation testing results on SRAM Cypress CY62256NLL. The top figure is the testing result on stable bits generated using neighbor analysis, while the bottom one is tested on stable bits produced by data remanence approach. Index on x-axis refers to two different voltages, e.g. 4.5-4.6 means the fractional hamming distance between enrollment on voltage 4.5V and voltage 4.6V.

0.185 seconds, while strong 0's are calculated when 0.27 seconds are used as the power down period. The difference between power down period during generation of strong 1's and strong 0's is because the number of 1's that flipped fast are more compared to 0's. This is also related to the 0's and 1's distribution during normal initialization (0's count for 30% and 1's filled 70%). Similar to the previous algorithm, the stability of bits produced by using this algorithm is also not good. The worst change happens when 8 days is used as the time interval between testing, showing as many as 2328 bits (HD_{intra} 49.93%) and also introduce an overlap between HD_{intra} and HD_{inter}.

Cypress CY62256NLL

- Neighbor Stability Analysis
  To get 4662 bits, there are six categories included; rank similar or higher than 16 with 712 bits, rank 15 with 350 bits, rank 14 with 502 bits, 726 bits of rank 13, 1104 bits of rank 12, and 1268 bits of rank 11. Under the voltage and time interval variation, the stable bits generated using neighbor stability analysis show decent reliability by having maximum 197 changing bits (HD_{intra} 4.23%) when the data is gathered on voltage 4.8V and 5V.

- Data Remanence Approach
  Unlike SRAM 23LC1024, power down period when enrolling strong 1's and 0's on CY62256NLL is not different. To get 4662 stable bits, both are enrolled using power down period 0.34 seconds. During the voltage and time interval variation, the stable bits produced by using algorithm also shows a promising result. It only accounts for maximum 73 bits difference (HD_{intra} 1.56%).

Stability Test Conclusion

Based on these results, SRAM Cypress CY62256NLL is shown to be a reliable SRAM candidate for PUF due to its well distribution of 1's 0's inside its memory and small variance when tested on various voltage and time interval between enrollment, especially the stable bits produced by data remanence analysis which has HD_{intra} less than 2% on any testing. If Cypress CY62256NLL is used as the root-of-trust to produce PUF-generated key, the key is ensured to always have the same value since the key generation scheme can tolerate up to 12.7% while the error rate of stable bits of Cypress CY62256NLL produced by data remanence algorithm is always less than 2%. Sadly, the other SRAM, SRAM Microchip 23LC1024, has displayed a poor performance to be eligible as a PUF candidate. Unbalanced 1's and 0's distribution and large HD_{intra} when the stable bits are tested (larger than the maximum error capability of the key generation scheme, and even introduces an overlap between $\mathrm{HD}_{\mathrm{intra}}$ and $\mathrm{HD}_{\mathrm{inter}}$) are two main reasons why this SRAM is not recommended to use as a PUF candidate.

These different results between two types of SRAMs lead us to a thinking that the SRAM size and the technology used in SRAM manufacturing affects a lot of SRAM quality as a PUF candidate. For example, Cypress CY62256NLL has significantly larger size than Microchip 23LC1024 (a rough approximation results in 13.38 times larger). Cypress CY62256NLL also has a smaller capacity (256 kbit) than Microchip 23LC1024 (1024 kbit). In addition, Cypress CY62256NLL is produced using an older technology (90nm) compared to Microchip 23LC1024 which has a higher chance to be produced using a newer technology since it has a much smaller size but larger memory size than Cypress CY62256NLL (there is no information on manufacturing technology used in the production on their websites and the Microchip 23LC1024 manual descriptions). From these explanations, we can conclude that Cypress CY62256NLL has less density than Microchip 23LC1024. These reasons lead us to a confirmation of density effects explained in Section 2.6.3 which says the more dense an SRAM, the more environments affect the performance of the SRAM. But does it mean that SRAM PUF cannot be produced using an SRAM with a high density? This seems untrue due to some SRAM PUF references mentioned a newer technology in their PUF constructions, e.g. Cortez et. al. in [75] use SRAMs which produced using 32nm and 45nm (sadly, there is no information on the type and manufacturer of their tested SRAMs). Moreover, as mentioned in Section 2.8, the experiments done in [65] shown that an older manufacturing technology does not always produced a more stable SRAM (Cypress CY7C15632KV18 (65nm) is more stable than Virage HP ASAP SP ULP 32-bit (90nm), even though the most stable is IDT 71V416S15PHI which produced using 180nm technology). Furthermore, since every company always has their own way of dealing with noises introduced by high density level, we cannot conclude that high density level always lead to low quality of an SRAM as a PUF candidate. Now, this lead to another questions, what is the main criteria if an off-the-shelf SRAM is going to be used a PUF candidate? Should we trust specific company such as Cypress and mistrust another company like Microchip? Or do we need to look into specific product to determine whether an SRAM is suitable for a PUF component? Should we always prefer SRAMs with less density? We suggest the communities and the academics to study these problems further.

Another conclusion that can be retrieved is that data remanence analysis is proven to be a better bit selection algorithm than neighbor analysis which also confirms similar claim by Muqing et. al. [55]. Futhermore, based on this outcome, further testing shown below are only done on SRAM Cypress

CY62256NLL and the stable bits used are generated using the data remanence algorithm.

4.5 Testing on Bits Locations as A Challenge

In this section, the testing results on our proposed PUF challenge is presented. As mentioned in the previous chapter, bits locations is selected as the PUF challenge in our application. The test was done on SRAM Cypress CY62256NLL. Cypress CY62256NLL itself has a capacity to store 262144 bits. The number of bits required in a challenge is 2331 bits (the length of the bits required to generate 256 bits key when using scheme shown in Figure 3.2). Using Equation 3.1 explained in previous chapter, there are $P(262144, 2331) = \frac{262144!}{(262144-2331)!} \approx 10^{12626}$ possible combinations.

The selected experiment to test this challenge is by calculating the HD_{inter} among five SRAMs. Figure 4.14 shows the result of this experiment. As shown in that figure, the HD_{inter} is ranging between 35.26% until 46.93%, with average 42.08%. Since there is no overlap between HD_{intra} (shown on Section 4.4.3) and HD_{inter}, this result shows that using bits locations as a challenge is sufficient to distinguish an SRAM from another.

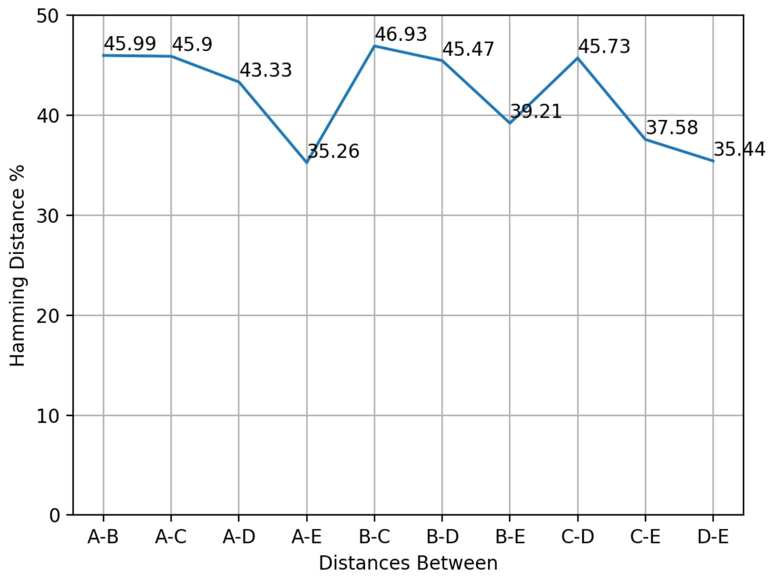


Figure 4.14: HD_{inter} among five SRAMs Cypress CY62256NLL.

4.6 Complete Enrollment and Reconstruction Scheme

Based on the experiment results shown before, we construct a complete enrollment and reconstruction scheme. The enrollment scheme has a goal to create challenge and helper data which will be used in our proposed secure data and key storage scheme (further explanation is available on Section 3.4). The reconstruction scheme has a function to reconstruct the PUF-generated key. Details on the key reconstruction can be seen on Figure 3.1 and Figure 3.2 in previous chapter.

Similar to the automatic profiling system, the enrollment scheme also consists of a PC, act as a master, and an Arduino connected to an external SRAM component which acts as a slave. The PC side will run Python codes while Arduino side requires Arduino codes. Our complete enrollment scheme is shown in Figure 4.15. We also present Figure 4.16 to show how to connect an Arduino Mega 2560, an SRAM Cypress CY62256NLL and a microSD.

The enrollment scheme starts by locating stable bits using bit selection algorithm. The chosen bit selection algorithm is data remanence analysis due to its better result and shorter time needed compared to neighbor analysis. Using this algorithm, we detect the position of 4662 stable bits. Afterwards, these stable bits are shuffled to form a set of 2331 bits locations which will be used as the PUF challenge. Using Equation 3.1 explained in previous chapter, there are $P(4662, 2331) = \frac{4662!}{(4662-2331)!} \approx 8.97 \times 10^{8240}$ possible stable bits combinations. The process continues with creating the helper data based on the PUF challenge. The enrollment scheme ends with storing the helper data and the PUF challenge to a microSD. Later, if one wants to reconstruct the PUF-generated key from the helper data and the challenge, he only requires an Arduino, no PC is needed (the Arduino codes used for reconstructing the PUF-generated key is different from the Arduino codes for testing and performing the enrollment scheme when it act as a slave).

4.7 Procedure on Developing SRAM PUF-Based Applications

After presenting the complete enrollment scheme, we also come up with a procedure to develop SRAM PUF-based applications using any off-the-shelf SRAM. Similar to the complete enrollment scheme, this procedure scheme also requires a PC, acting as a master, and an Arduino connected to an external SRAM component which acts as a slave. The PC side will run Python code. To reconstruct the PUF-generated key, similar to the enrollment scheme, one also only need an Arduino, no PC is needed. This procedure will check whether the quality of the off-the-shelf SRAM is suf-

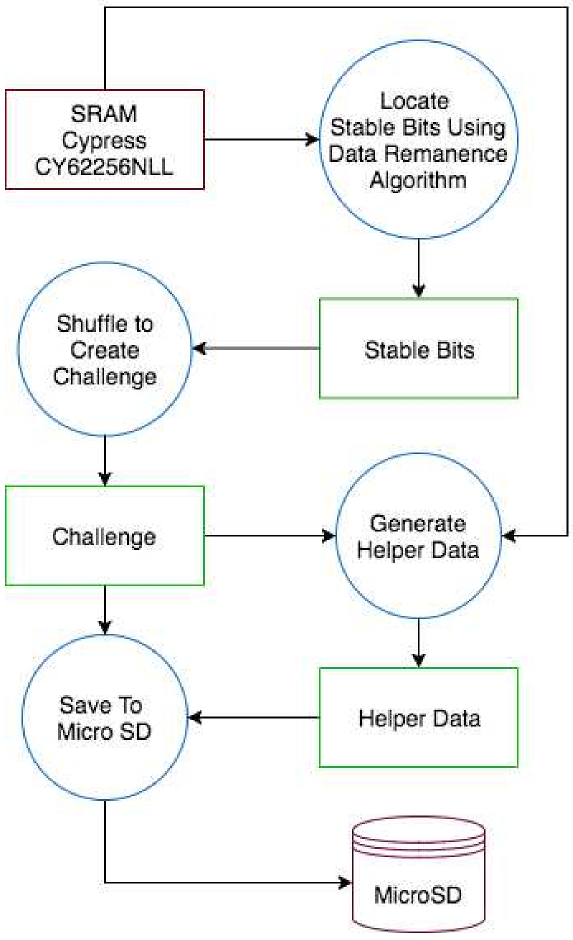


Figure 4.15: Complete enrollment scheme using SRAM Cypress CY62256NLL, data remanence algorithm and stable bits locations as the PUF challenge. At the end of the enrollment scheme, the challenge and helper data will be generated and saved in a microSD.

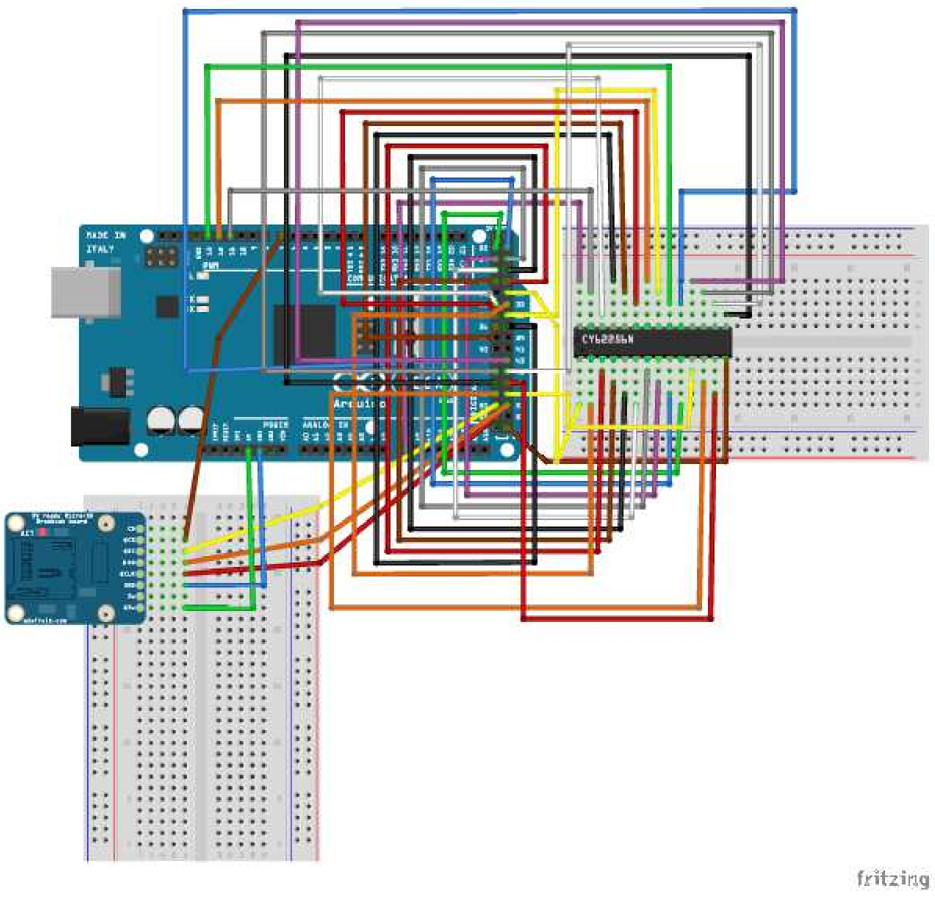


Figure 4.16: An illustration on how to connect an Arduino Mega 2560 with an SRAM Cypress CY62256NLL and a microSD.

ficient as a PUF root-of-trust or not. In addition, the procedure also generates the helper data and the challenge of the SRAM which can be used to reconstruct PUF-generated key. Using this PUF-generated key, anyone can build their applications. For example, this PUF-generated key can be utilized as the root-of-trust in an authentication application. Below are the steps defined in this procedure:

- Step 1: *Test the off-the-shelf SRAM quality as a PUF component.*
  This step can be started by getting any off-the-shelf SRAM from the market. Ensure that the Arduino and the PC are able to communicate perfectly with off-the-shelf SRAM. Afterwards, locate the stable bits using data remanence algorithm. Then, test the stable bits on various conditions. If the stable bits show an acceptable result, continue using this SRAM. Otherwise, abandon and try another off-the-shelf SRAM. In addition, also ensure there is no overlap between HD_{inter} and HD_{intra} to enable unique identification on the off-the-shelf SRAM.

- Step 2: *Use enrollment-reconstruction mechanism which will be able to create a PUF-generated key.*
  Using the enrollment scheme shown in Section 4.6, one can generate the challenge and the helper data. Both the helper data and the challenge is unique to the tested SRAM component. Later, using the challenge and the helper data, one will be able to reconstruct the PUF-generated key.

- Step 3: *Develop a PUF-based application using the PUF-generated key.*
  The PUF-generated key can be applied as a root-of-trust in any application.

4.8 Testing on Secure Data and Key Storage Scheme

To check the validity of the proposed data and key storage scheme, several testings are performed. First, the final key generated using HMAC SHA3 is checked. Afterwards, the result of the encryption and decryption using the final key is also tested. Last, the time required in the scheme is also measured. During the time measurement, the stage is divided into multiple stages to ease the analysis.

To check the validity of the final key, a comparison between the generated final key and an online HMAC SHA3 calculator [76] is performed. As a reminder, the key for HMAC function is the PUF generated key and the message input for the HMAC is the user's password. Below is the testing result:

- PUF generated key: d20f5656bf436516cd0f3d2e734851dc537df518 97484128ccae67ee1310f69b
  - user's password: 70617373776f7264
    final key: 084536fcb3135af89e1e32d423156511f13e52246acaa 591b1d4115666727814
    valid: yes
  - user's password: 6b6f6e746f6c6b61626568
    final key: 1d1e467224d72c81ede61fcd5d1ac10535f3ebafa6e9f 0d5086e6086a30787c7
    valid: yes
  - user's password: 71776572747975696f706173646667686a6b6c 7a786376626e6d
    final key: dda21605fc56b55659cffdf57f5453a9e380aa7bd78fe5 2b7dc64ff4515ff4a0
    valid: yes
- PUF generated key: 35e2f312bd28a36a359eb1a1e37f212d17da41a5 b17cb2c642f5fd8e42bbd4f0
  - user's password: 70617373776f7264
    final key: c2892f1b1d52d59549591d410a40527b265b91d444d 2032f28ce7374f7246152
    valid: yes
  - user's password: 6b6f6e746f6c6b61626568
    final key: ec85915ae65f3e5141128a520327c4d5cd3119cb6769f ddd948d3061dfb6fed9
    valid: yes
  - user's password: 71776572747975696f706173646667686a6b6c 7a786376626e6d
    final key: da9e28f76754dcd4946c1343a3dd8550338d98e46d3a 11e09f903204044ac9c7
    valid: yes

After checking the validity of the final key, a testing on encryption and decryption using the final key is performed. The input data on this testing is users key. To check the validity of the ciphertext, an online encryption calculator [77] is utilized. The ciphertext result of the encryption process will be used as the input for the decryption test. If the decryption result is similar with the users key, then both encryption and decryption process in this scheme is valid.

- user's key: 70617373776f726470617373776f726470617373776f726 470617373776f7264

- final key: e2e1d413041797bdd88509a36333ff4488c02fad7dfb8 e7f4c490ba3b532dbf0
  IV: 454de46011532218d7651f13b719c74f
  ciphertext: 908d70d57d70de5f99879fe686df1f3cdf42ae781951 a681dadad3e264e693e6
  ciphertext validity: yes
  decryption result: 70617373776f726470617373776f726470617 373776f726470617373776f7264
  decryption validity: yes
- final key: 5c2c9c88bf7f8af53b74c200d28d2d5f2126b319ad9bf 5153af57f8509d40855
  IV: 0e6ea6ddb70ecb04cef60d16f8574480
  ciphertext: 28306abb964b77d736c40cb4f9eaf9e950928e46379 eac789a822746611dcdcd
  ciphertext validity: yes
  decryption result: 70617373776f726470617373776f726470617 373776f726470617373776f7264
  decryption validity: yes
- final key: ec41183c27222149f1f7e7d0c801807b76482532e186 302c49a4428104d394e1
  IV: 199a195d30b54308030c441551b12ba9
  ciphertext: b4fc405e19a1a00ee7c7c97946a300f02d61bb64eba9 f72517bd29e8f59e6ce5
  ciphertext validity: yes
  decryption result: 70617373776f726470617373776f726470617 373776f726470617373776f7264
  decryption validity: yes
- final key: 95a68d46b3d4477053424895822 09f48cf654360b7af 96d803fe4bb7d5596bfe
  IV: 9e5b0a358f8b4a19931787bd2b371407
  ciphertext: eac69afe87a1757dd5bbfcb4f7cce0f5f8cec76860583 ed10ecfc7107f575d29
  ciphertext validity: yes
  decryption result: 70617373776f726470617373776f726470617 373776f726470617373776f7264
  decryption validity: yes
- final key: d62ae23a0a45fc94755d0e1523fe5d908b116fda239a9 4852d849c631aec8024
  IV: 199a195d30b54308030c441551b12ba9
  ciphertext: 25f59967dcdd36f24761497fe310176b2282ecaef4d5 f9c22f95410161057ebe
  ciphertext validity: yes

decryption result: 70617373776f726470617373776f726470617
373776f726470617373776f7264
decryption validity: yes

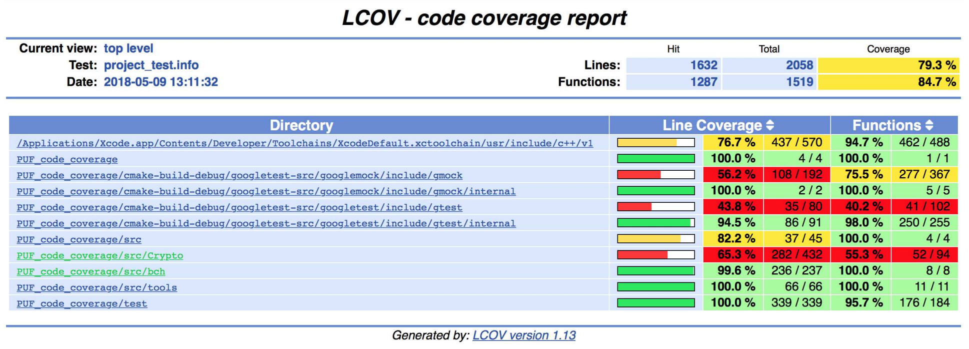
- final key: b7dc7382a1d6cb47970446797e8ab45385b0b23d8b3f
bfea05447004b8ebd17c
IV: 24c78cdea85dba0c38227b15ab0b12d2
ciphertext: 876f0510c5cc2157affbece024494db6718e9b1ad1b9
0233782926919acf8409
ciphertext validity: yes
decryption result: 70617373776f726470617373776f726470617
373776f726470617373776f7264
decryption validity: yes

Measurement of time required in this scheme is also done. During the time measurement, the scheme is divided into eight stages. Stage one is on the initialization of the libraries required to access SRAM Cypress CY62256NLL and microSD. Stage two is when the challenge and the helper data are loaded from microSD. The third one is calculated when reconstructing the PUF key. Next stage is during the derivation of the final key (derived from user's password and PUF-generated key). Stage five and stage six refers to the processes of encryption and saving ciphertext to microSD. Stage seven and stage eight refers to the procedure of reading ciphertext from microSD and the decryption process (reconstructing the user's key). The measurement result can be seen on Table 4.1. It can be seen that the longest time required is when loading the challenge and the helper data from the microSD (stage 2), followed by the initialization stage (stage 1). Due to this significant time required, a further optimization on accessing data from microSD is suggested in possible future works.

Table 4.1: Time measurement of the secure data and key storage scheme in ms.

No	**Stage 1**	**Stage 2**	**Stage 3**	**Stage 4**
1	1022.66	2205.25	978.15	33.57
2	1022.65	2205.24	974.39	33.57
3	1022.63	2205.25	981.27	33.57
Average	1022.65	2205.25	977.94	33.57
No	**Stage 5**	**Stage 6**	**Stage 7**	**Stage 8**
1	0.84	39.96	13.02	1.72
2	0.85	39.88	13.02	1.71
3	0.84	39.78	13.01	1.72
Average	0.84	39.87	13.02	1.71

To ensure the functionality of the system, we also constructed test cases for the source code. The testing is only performed on the functionality of the system which does not has a direct interaction with hardware (e.g. code to read microSD and SRAM). This decision is taken because to ensure the code is properly working, the hardware has to be in a proper condition, while checking the hardware condition is sometimes problematic and cannot be automated. The quality of the test cases is shown by code coverage which generated using GCOV and LCOV. Figure 4.17 shows the code coverage result. As seen there, there are directories where the line coverage is not 100% covered. The reasons are some directories are just library to enable code testing (folder 'gmock' and folder 'gtest', both are libraries created by Google which required to enable the testing) and one directory is a part of LLVM compiler infrastructure (folder 'v1'). Another directory which is not fully covered is 'Crypto', this folder is a library required to do the encryption, decryption, and HMAC but provides more functional than the required for constructing the system. Thus, the test is only done on the functionality that is actually needed by the system which leads to a not full percentage of code coverage.



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level				
Test:	project_test.info	Lines:	1632	2058	79.3 %
Date:	2018-05-09 13:11:32	Functions:	1287	1519	84.7 %

Directory	Line Coverage ⇕			Functions ⇕	
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1		76.7 %	437 / 570	94.7 %	462 / 488
PUF_code_coverage		100.0 %	4 / 4	100.0 %	1 / 1
PUF_code_coverage/cmake-build-debug/googletest-src/googlemock/include/gmock		56.2 %	108 / 192	75.5 %	277 / 367
PUF_code_coverage/cmake-build-debug/googletest-src/googlemock/include/gmock/internal		100.0 %	2 / 2	100.0 %	5 / 5
PUF_code_coverage/cmake-build-debug/googletest-src/googletest/include/gtest		43.8 %	35 / 80	40.2 %	41 / 102
PUF_code_coverage/cmake-build-debug/googletest-src/googletest/include/gtest/internal		94.5 %	86 / 91	98.0 %	250 / 255
PUF_code_coverage/src		82.2 %	37 / 45	100.0 %	4 / 4
PUF_code_coverage/src/Crypto		65.3 %	282 / 432	55.3 %	52 / 94
PUF_code_coverage/src/bch		99.6 %	236 / 237	100.0 %	8 / 8
PUF_code_coverage/src/tools		100.0 %	66 / 66	100.0 %	11 / 11
PUF_code_coverage/test		100.0 %	339 / 339	95.7 %	176 / 184

Generated by: LCOV version 1.13

Figure 4.17: Code coverage of the constructed project.

4.9 Concluding Experiment with Cybercurrency

As the final experiment of this thesis, we present a demo of storing a private key of a cybercurrency. We believes this proves the usefulness and viability of this work for realistic use-cases. The chosen cybercurrency in this demo is Bitcoin. In Bitcoin, the private key has a length of 256-bit or 32 bytes [78]. The experiments starts by performing an enrollment on an Arduino board with an SRAM Cypress CY62256NLL and a microSD connected to it, resulting in challenge and helper data which store in the microSD. Using the produced challenge and helper data, user creates a final key which derived from the PUF-generated key and user's password. The final key is utilized to encrypt a Bitcoin key, then the ciphertext is stored in microSD. Afterwards, the Arduino is turned off. Later, the microSD and the SRAM are transferred to another Arduino board. The new Arduino board is powered on, then it is used to reconstruct the final key by inputing the correct user's password. Finally, the reconstructed final key is applied to the ciphertext which is loaded from the microSD. The Bitcoin key storing experiment is considered successful if the result of decryption is the same as the Bitcoin key. Moreover, we also shows that the Bitcoin key will not be reconstructed successfully if user's password is incorrect or the SRAM is not similar with the one that use to encrypt the Bitcoin key.

This experiment was done on five SRAMs Cypress CY62256NLL which can be identified by having index 'A', 'B', 'C', 'D', and E. The result of this experiment is shown on Appendix A. These figures show that the stored / secured Bitcoin key can only be reconstructed using a correct user's password and the exact SRAM that used during the storing (encryption) stage. If the SRAM is not similar with the one used for the encryption stage or the input password is inaccurate, the Bitcoin key cannot be reconstructed to the actual one. Based on these result, we conclude that the constructed data and key storage scheme is secure and successfully built.

4.10 Conclusion

In this chapter, we provide explanations of several experiment setups and results. We start by presenting two chosen SRAMs that used in experiments; Microchip 23LC1024 and Cypress CY62256NLL. Both are tested on the effect of voltage variations regarding the HD_{intra}, HD_{inter}, distribution of 1's and 0's. Afterwards, the testing results on two bit selection algorithms (neighbor analysis and data remanence approach) and the reliability of the stable bits produced by these algorithms are displayed. From these experiments, we concluded that the Microchip 23LC1024 is unreliable to be a PUF candidate while Cypress CY62256NLL has a solid ground to be the root-of-trust of PUF. We also able to determine that data remanence approach can produce more stable bits compared to neighbor analysis. The chapter continues with examination on our proposed PUF challenge. Since Cypress CY62256NLL has a capacity of 262144 bits and the required bits for PUF generation is 2331 bits, there are $P(262144, 2331) = \frac{262144!}{(262144-2331)!} \approx 10^{12626}$ possible challenges. We also display our complete enrollment scheme and the procedure on developing SRAM PUF-based applications using any off-the-shelf SRAM. Moreover, we also propose a procedure to develop SRAM PUF-based applications using any off-the-shelf SRAM. The procedure consists of three main steps; test the off-the-shelf SRAM quality to be a PUF component, create a PUF-generated key using enrollment-reconstruction mechanism, and develop any PUF-based application utilizing the PUF-generated key. Next, testing on the designed secure data and key storage scheme is shown. Last, experiment outcomes on storing Bitcoin private key conclude this chapter.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis starts by showing the potential of using SRAM PUF as a secure way to protect our key and data. Embraced with a bright prospect, it is unfortunate that the development of PUF in the real world seems to lack of public involvement. Currently available solutions are usually locked to specific entities, such as companies or universities. There is no open source project available for tech enthusiast to embrace this amazing technology. Here, we introduce the first open source project to develop software-based SRAM PUF technology using off-the-shelf SRAM.

As mentioned in Chapter 1, this thesis' problem statement says "How to develop an open-source secure data and key storage scheme using off-the-shelf SRAM component and software-based SRAM PUF technology?". Based on the experiments, the answer of this question can be explained in few steps. The steps itself are quite similar with the one explained in Section 4.7. First, one should test the off-the-shelf SRAM quality to be a PUF component. If passed, the procedure continues to the next step which consists of enrollment-reconstruction mechanism which will be able to create a PUF-generated key. Last, using the PUF-generated key, we develop our secure data and key storage scheme. To secure data/key, we decided to encrypt it using AES-256 with a *final key* as the encryption key. The final key itself is derived using HMAC-SHA3 256 as the KDF (key derivation function) by combining PUF-generated key and user password. We consider this scheme is secure because the only-way someone can access the encrypted data is by having the SRAM and knowing the user password. Thus, the data and the key protected by the scheme is still safe even though the PUF device is lost. In addition, ensure all the library required in the constructed system is open-source. Moreover, software-based SRAM PUF technology is achieved by using off-the-shelf SRAM (requires no added hardware design) and constructing the whole system using Arduino and Python codes.

In problem statement section, there are also two goals defined. First is "to devise a secure data and key storage scheme based on SRAM PUF technology. The data and the key protected by the scheme has to be safe even though the PUF device is lost. Moreover, the scheme should work using off-the-shelf SRAM." This goal is achieved and the explanation can be found in previous paragraph.

From the first goal, several sub-questions are also arised. First question is "Can we build SRAM PUF using off-the-shelf SRAM?". The answer is yes, we can. We show that the stable bits of SRAM Cypress CY62256NLL which located using data remanence algorithm has a sufficient reliability to be a root-of-trust of SRAM PUF.

Second question is "If it is possible, what characteristics need to be fulfilled by off-the-shelf SRAM to be eligible as a PUF candidate?". Unfortunately, we cannot answer this question. As shown in Chapter 4, we also do experiment on another off-the-shelf SRAM called Microchip 23LC1024. This SRAM show an unsatisfying result to be a PUF candidate. We believe that the SRAM size and the technology used in SRAM manufacturing affects a lot of SRAM quality as a PUF candidate. For example, Cypress CY62256NLL has significantly larger size than Microchip 23LC1024 (a rough approximation results in 13.38 times larger). Cypress CY62256NLL also has a smaller capacity (256 kbit) than Microchip 23LC1024 (1024 kbit). In addition, Cypress CY62256NLL is produced using an older technology (90nm) compared to Microchip 23LC1024 which has a higher chance to be produced using a newer technology since it has a much smaller size but larger memory size than Cypress CY62256NLL (there is no information on manufacturing technology used in the production on their websites and the Microchip 23LC1024 manual descriptions). From these explanations, we can conclude that Cypress CY62256NLL has less density than Microchip 23LC1024. These reasons lead us to a confirmation of density effects explained in Section 2.6.3 which says the more dense an SRAM, the more environments affect the performance of the SRAM. But does it mean that SRAM PUF cannot be produced using an SRAM with a high density? This seems untrue due to some SRAM PUF references mentioned a newer technology in their PUF constructions, e.g. Cortez et. al. in [75] use SRAMs which produced using 32nm and 45nm (sadly, there is no information on the type and manufacturer of their tested SRAMs). Moreover, as mentioned in Section 2.8, the experiments done in [65] shown that an older manufacturing technology does not always produced a more stable SRAM (Cypress CY7C15632KV18 (65nm) is more stable than Virage HP ASAP SP ULP 32-bit (90nm), even though the most stable is IDT 71V416S15PHI which produced using 180nm technology). Furthermore, since every company always has their own way of dealing with noises introduced by high density level, we cannot conclude that high density level always lead to low quality of an SRAM as a PUF candidate. This open conclusion lead us to other questions. Should we trust specific company such as Cypress and mistrust another company like Microchip? Or do we need to look into specific product to determine whether an SRAM is suitable for a PUF component? Should we always prefer SRAMs with less density? We suggest the communities and the academics to study these problems further.

The second goal defined in the problem statement is "create a sharing ecosystem for the evolution of our data and key storage scheme. The ecosystem should be easily accessed and understood to encourage the academics and commercial parties to use and develop the ecosystem together." We achieved this goal by providing all our codes with explanations on how to use them in a Github repository [23].

In addition, an idea to create numerous CRPs using SRAM PUF is also proposed here. Using a collection of bits as a challenge, the stable bits are permutated among themselves to create a challenge which has a tremendous number of possibilities.

Moreover, as the final experiment of this thesis, we present a demo of storing a private key of Bitcoin. We shows that the Bitcoin key will not be reconstructed successfully if user's password is incorrect or the SRAM is not similar with the one that use to encrypt the Bitcoin key.

5.2 Future Work

In this section, two major parts on possible future work are presented. The first part is about possible experiments to do on off-the-shelf SRAMs and the next part is related to possible developments on our secure data and key storage scheme. Explanations of these two will be provided below.

Possible Experiments on Off-The-Shelf SRAMs

In this thesis, the SRAM testing is only done on the effect of time interval between data retrieval and voltage variation. We believe another testing on temperature and aging is required to ensure whether SRAM Cypress CY62256NLL is indeed a capable candidate for SRAM PUF. The capability to test on temperature and the aging effect is suggested to be included as an addition of our automated enrollment system.

In addition, we also encouraged others to test other types of SRAMs to enrich the knowledge of possible off-the-shelf SRAM as a PUF candidate and to check if a high-density level always leads to a poor performance for an SRAM to be a PUF candidate. Doing testing on other types of SRAMs can also confirm whether a product from specific is qualified as a PUF root-of-trust or not.

Improvement on Secure Data and Key Storage Scheme

As mentioned in Section 4.8, during the time measurement of our proposed key storage scheme, two procedures which spend significant time is the process of reading challenge from microSD and the initialization stages. We suggest to further optimize these stages to give a better and faster performance.

This thesis only presents an idea to secure user's data using symmetric encryption. To see similar application but using asymmetric encryption concept, one should look further to the thesis done by Akhundov [64]. He presents a public key infrastructure (PKI) concept using the PUF-generated key as the root of trust. A possible integration between our work and his work is combining our 'final' key into his construction as a root of trust.

Moreover, our secure data and key storage scheme is only designed to work offline. We believe by making it works in an online scenario will lead to more usable applications in real life. The first step we suggest on evolving it to be an online scheme is by providing the Arduino with an internet connection and by storing the helper data and the challenge in the cloud infrastructure. This step will reduce the necessity for the Arduino to always connected to a microSD. To reconstruct the PUF-generated key, Arduino will just have to get the challenge and the helper data from the cloud. We also advise to do extensive security analysis if it is decided to work online since the risks in an online environment are numerous.

In addition, our idea of using user's password and the PUF-generated key is not the highest level of security in multi-factor authentication. As mentioned in Section 2.4, the most secure multi-factor authentication can be achieved when all three factors are combined together; knowledge, possession, and inherence. Since there are only two factors utilize (knowledge and possession) in this thesis' proposed secure data and key storage scheme, an addition of inherence factor when generating the final key can increase the security level. As mentioned in Section 1.2, biometric-based authentication and PUF are utilized to secure self-sovereign identity in Pouwelse and de Vos's proposed technology stack during trust creation in blockchain era. A further read on their article mentioned that there is a working prototype of fingerprint authentication using a smartphone camera. Since that project and our work share the same principle, open source and open ecosystem, we suggest integrating this fingerprint authentication into our proposed scheme to enable an even higher level of security.

Bibliography

[1] World economic forum - a blueprint for digital identity. http://www3.weforum.org/docs/WEF_A_Blueprint_for_Digital_Identity.pdf. [Online; accessed 16-March-2018].

[2] G R Blakley and David Chaum, editors. *Proceedings of CRYPTO 84 on Advances in Cryptology*, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[3] What is "sovereign source authority"? — the moxy tongue. http://http://www.moxytongue.com/2012/02/what-is-sovereign-source-authority.html. [Online; accessed 19-February-2018].

[4] Johan Pouwelse, Andr De Kok, Joost Fleuren, Peter Hoogendoorn, Raynor Vliegendhart, and Martijn De Vos. Laws for creating trust in the blockchain age. *European Property Law Journal*, 6(3), Dec 2017.

[5] Eur-lex - 32016r0679 - en. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32016R0679. [Online; accessed 06-March-2018].

[6] Data privacy vs. data protection. https://blog.ipswitch.com/data-privacy-vs-data-protection. [Online; accessed 16-March-2018].

[7] Sergei Skorobogatov. Physical attacks and tamper resistance. *Introduction to Hardware Security and Trust*, page 143173, Aug 2011.

[8] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.

[9] C. H. Chang, Y. Zheng, and L. Zhang. A retrospective and a look forward: Fifteen years of physical unclonable function advancement. *IEEE Circuits and Systems Magazine*, 17(3):32–62, thirdquarter 2017.

[10] Pim Tuyls. *"Security with Noisy Data: On Private Biometrics, Secure Key Storage and Anti-Counterfeiting"*. Springer, 2010.

[11] Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, and Pim Tuyls. Fpga intrinsic pufs and their use for ip protection. *Cryptographic Hardware and Embedded Systems - CHES 2007 Lecture Notes in Computer Science*, page 6380.

[12] Google patents. `https://patents.google.com/?q=sram&q=puf&oq=srampuf`. [Online; accessed 06-March-2018].

[13] Google scholar. `https://scholar.google.nl/scholar?as_vis=1&q=srampuf&hl=en&as_sdt=1,5`. [Online; accessed 06-March-2018].

[14] Broadkey - intrinsic id — iot security. `https://www.intrinsic-id.com/products/broadkey/`. [Online; accessed 06-March-2018].

[15] Polarfire evaluation kit. `https://www.microsemi.com/products/fpga-soc/design-resources/dev-kits/polarfire/polarfire-eval-kit`. [Online; accessed 06-March-2018].

[16] Pim Tuyls and Boris Škorić. *Strong Authentication with Physical Unclonable Functions*, pages 133–148. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[17] Jeroen Delvaux, Roel Peeters, Dawu Gu, and Ingrid Verbauwhede. A survey on lightweight entity authentication with strong pufs. *ACM Comput. Surv.*, 48(2):26:1–26:42, October 2015.

[18] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th Annual Design Automation Conference*, DAC ’07, pages 9–14, New York, NY, USA, 2007. ACM.

[19] Keith B. Frikken, Marina Blanton, and Mikhail J. Atallah. Robust authentication using physically unclonable functions. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio A. Ardagna, editors, *Information Security*, pages 262–277, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[20] Heike Busch, Stefan Katzenbeisser, and Paul Baecher. Puf-based authentication protocols – revisited. In Heung Youl Youm and Moti Yung, editors, *Information Security Applications*, pages 296–308, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[21] Amanda C. Davi Resende, Karina Mochetti, and Diego F. Aranha. Puf-based mutual multifactor entity and transaction authentication for secure banking. In Tim Güneysu, Gregor Leander, and Amir Moradi, editors, *Lightweight Cryptography for Security and Privacy*, pages 77–96, Cham, 2016. Springer International Publishing.

[22] Roel Maes, Anthony Van Herrewege, and Ingrid Verbauwhede. Pufky: A fully functional puf-based cryptographic key generator. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 302–319, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[23] Github - tribler/software-based-puf/ the first open source software-based puf. `https://github.com/Tribler/software-based-PUF`. [Online; accessed 15-April-2018].

[24] H. X. Mel and Doris Baker. *Cryptography decrypted.* Addison-Wesley, 2003.

[25] C. Soanes and A. Stevenson. *Concise Oxford English Dictionary*. Concise Oxford English Dictionary. Oxford University Press, 2008.

[26] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[27] Nigel P. Smart. *Cryptography Made Simple*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[28] Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 631–648, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[29] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 313–314, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[30] Keccak team - the sponge and duplex constructions. `https://keccak.team/sponge_duplex.html`. [Online; accessed 20-Jan-2018].

[31] N.I.N.I.S. Technology. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions: FiPS PUB 202*. CreateSpace Independent Publishing Platform, 2015.

[32] Chiara Galdi, Michele Nappi, Jean-Luc Dugelay, and Yong Yu. Exploring new authentication protocols for sensitive data protection on smartphones. *IEEE Communications Magazine*, 56:136–142, 2018.

[33] U. Rhrmair and D. E. Holcomb. Pufs at a glance. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[34] Roel Maes. *Physically Unclonable Functions Constructions, Properties and Applications*. Springer Berlin, 2016.

[35] Roel Maes and Ingrid Verbauwhede. *Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions*, pages 3–37. 10 2010.

[36] Hamming distance — practice problems — hackerearth. `https://www.hackerearth.com/problem/algorithm/hamming-distance-1/`. [Online; accessed 10-Jan-2018].

[37] M. Cortez, A. Dargar, S. Hamdioui, and G. J. Schrijen. Modeling sram start-up behavior for physical unclonable functions. In *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6, Oct 2012.

[38] A. Maiti and P. Schaumont. The impact of aging on a physical unclonable function. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(9):1854–1864, Sept 2014.

[39] Christoph Bösch, Jorge Guajardo, Ahmad-Reza Sadeghi, Jamshid Shokrollahi, and Pim Tuyls. Efficient helper data key extractor on fpgas. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, pages 181–197, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[40] R. Maes, P. Tuyls, and I. Verbauwhede. A soft decision helper data algorithm for sram pufs. In *2009 IEEE International Symposium on Information Theory*, pages 2101–2105, June 2009.

[41] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 523–540, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[42] Renato Renner and Stefan Wolf. Simple and tight bounds for information reconciliation and privacy amplification. In Bimal Roy, editor,

Advances in Cryptology - ASIACRYPT 2005, pages 199–216, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[43] M. Taniguchi, M. Shiozaki, H. Kubo, and T. Fujino. A stable key generation from puf responses with a fuzzy extractor for cryptographic authentications. In *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pages 525–527, Oct 2013.

[44] Robert H. Morelos-Zaragoza. *The Art of Error Correcting Coding*. Wiley, 2006.

[45] Apurva Dargar. Modeling sram start-up behavior for physical unclonable functions. MSc thesis, Delft University of Technology, 2011.

[46] A. Garg and T. T. Kim. Design of sram puf with improved uniformity and reliability utilizing device aging effect. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1941–1944, June 2014.

[47] Xiaoxiao Wang and Mohammad Tehranipoor. Novel physical unclonable function with process and environmental variations. *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, 2010.

[48] Mohamed H. Abu-Rahma and Mohab Anis. *Variability in Nanometer Technologies and Impact on SRAM*, pages 5–47. Springer New York, New York, NY, 2013.

[49] Vikram G Rao and Hamid Mahmoodi. Analysis of reliability of flip-flops under transistor aging effects in nano-scale cmos technology. *2011 IEEE 29th International Conference on Computer Design (ICCD)*, 2011.

[50] Elie Maricau and Georges Gielen. *CMOS Reliability Overview*, pages 15–35. Springer New York, New York, NY, 2013.

[51] B. Kaczer, R. Degraeve, M. Rasras, K. van de Mieroop, P. J. Roussel, and G. Groeseneken. Impact of MOSFET gate oxide breakdown on digital circuit operation and reliability. *IEEE Transactions on Electron Devices*, 49:500–506, March 2002.

[52] B. C. Paul, Kunhyuk Kang, H. Kufluoglu, M. A. Alam, and K. Roy. Temporal performance degradation under nbti: Estimation and design for improved reliability of nanoscale circuits. In *Proceedings of the Design Automation Test in Europe Conference*, volume 1, pages 1–6, March 2006.

[53] P. Magnone, F. Crupi, N. Wils, R. Jain, H. Tuinhout, P. Andricciola, G. Giusi, and C. Fiegna. Impact of hot carriers on nmosfet variability in 45- and 65-nm cmos technologies. *IEEE Transactions on Electron Devices*, 58(8):2347–2353, Aug 2011.

[54] Kan Xiao, Md. Tauhidur Rahman, Domenic Forte, Yu Huang, Mei Su, and Mohammad Tehranipoor. Bit selection algorithm suitable for high-volume production of sram-puf. *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014.

[55] Muqing Liu, Chen Zhou, Qianying Tang, Keshab K. Parhi, and Chris H. Kim. A data remanence based approach to generate 100sram physical unclonable function. *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2017.

[56] Hyunho Kang, Yohei Hori, Toshihiro Katashita, Manabu Hagiwara, and Keiichi Iwamura. Performance analysis for puf data using fuzzy extractor. In Young-Sik Jeong, Young-Ho Park, Ching-Hsien (Robert) Hsu, and James J. (Jong Hyuk) Park, editors, *Ubiquitous Information Technologies and Applications*, pages 277–284, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[57] H. Kang, Y. Hori, T. Katashita, M. Hagiwara, and K. Iwamura. Cryptographic key generation from puf data using efficient fuzzy extractors. In *16th International Conference on Advanced Communication Technology*, pages 23–26, Feb 2014.

[58] Ari Juels and Martin Wattenberg. A fuzzy commitment scheme. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, CCS '99, pages 28–36, New York, NY, USA, 1999. ACM.

[59] L. Kusters, T. Ignatenko, F. M. J. Willems, R. Maes, E. van der Sluis, and G. Selimis. Security of helper data schemes for sram-puf in multiple enrollment scenarios. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 1803–1807, June 2017.

[60] Boris Skoric, Geert-Jan Schrijen, Pim Tuyls, Tanya Ignatenko, and Frans Willems. *Secure Key Storage with PUFs*, pages 269–292. Springer London, London, 2007.

[61] Pim Tuyls, Geert-Jan Schrijen, Boris Škorić, Jan van Geloven, Nynke Verhaegh, and Rob Wolters. Read-proof hardware from protective coatings. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 369–383, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[62] Anthony Van Herrewege, André Schaller, Stefan Katzenbeisser, and Ingrid Verbauwhede. Demo: Inherent pufs and secure prngs on commercial off-the-shelf microcontrollers. In *Proceedings of the 2013 ACM SIGSAC conference on Computer &#38; communications security*, CCS '13, pages 1333–1336, New York, NY, USA, 2013. ACM.

[63] Nikolaos Athanasios Anagnostopoulos, Stefan Katzenbeisser, Markus Rosenstihl, Andr Schaller, Sebastian Gabmeyer, and Tolga Arul. Low-temperature data remanence attacks against intrinsic sram pufs. Cryptology ePrint Archive, Report 2016/769, 2016. `https://eprint.iacr.org/2016/769`.

[64] Haji Akhundov. Design & development of public-key based authentication architecture for iot devices using puf. MSc thesis, Delft University of Technology, 2017.

[65] Geert-Jan Schrijen and Vincent van der Leest. Comparative analysis of sram memories used as puf primitives. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 1319–1324, San Jose, CA, USA, 2012. EDA Consortium.

[66] D. E. Holcomb, W. P. Burleson, and K. Fu. Power-up sram state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers*, 58(9):1198–1210, Sept 2009.

[67] Arduino - about us. `https://www.arduino.cc/en/Main/AboutUs`. [Online; accessed 20-Jan-2018].

[68] The error correcting codes (ecc) page. `http://www.eccpage.com`. [Online; accessed 09-November-2017].

[69] S Stanzione and Giuseppe Iannaccone. Silicon physical unclonable function resistant to a 10^25-trial brute force attack in 90 nm cmos. pages 116 – 117, 07 2009.

[70] Fermilab — science — inquiring minds — questions about physics. `http://www.fnal.gov/pub/science/inquiring/questions/atoms.html`. [Online; accessed 10-April-2018].

[71] Biaoshuai Tao and Hongjun Wu. Improving the biclique cryptanalysis of aes. In Ernest Foo and Douglas Stebila, editors, *Information Security and Privacy*, pages 39–56, Cham, 2015. Springer International Publishing.

[72] Donghoon Chang. Security evaluation report on sha-224, sha-512/224, sha-512/256, and the six sha-3 functions. `http://www.`

cryptrec.go.jp/estimation/techrep_id2403_2.pdf, March 2015. [Online; accessed 15-April-2018].

[73] 23lc1024 - memory - microcontrollers and processors. http://www.microchip.com/wwwproducts/en/23LC1024. [Online; accessed 15-Dec-2017].

[74] Cy62256nll-70pxc. http://www.cypress.com/part/cy62256nll-70pxc. [Online; accessed 15-Dec-2017].

[75] M. Cortez, S. Hamdioui, and R. Ishihara. Design dependent sram puf robustness analysis. In *2015 16th Latin-American Test Symposium (LATS)*, pages 1–6, March 2015.

[76] Hmac generator online hash encryption. https://www.liavaag.org/English/SHA-Generator/HMAC/. [Online; accessed 26-March-2018].

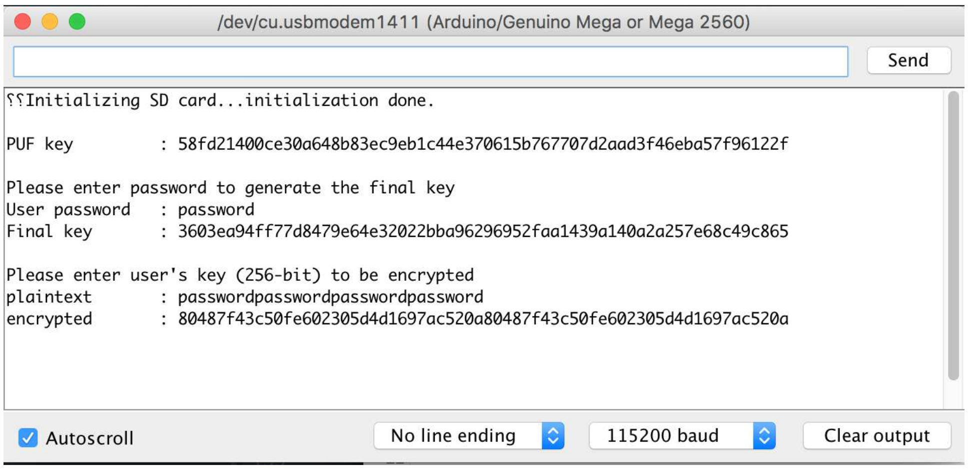
[77] Aes ctr online calculator. http://www.cryptogrium.com/aes-ctr.html. [Online; accessed 27-March-2018].

[78] Private key - bitcoin wiki. https://en.bitcoin.it/wiki/Private_key. [Online; accessed 12-April-2018].

Appendix A

Screenshot of Secure Data and Key Storage Scheme

Appendix body



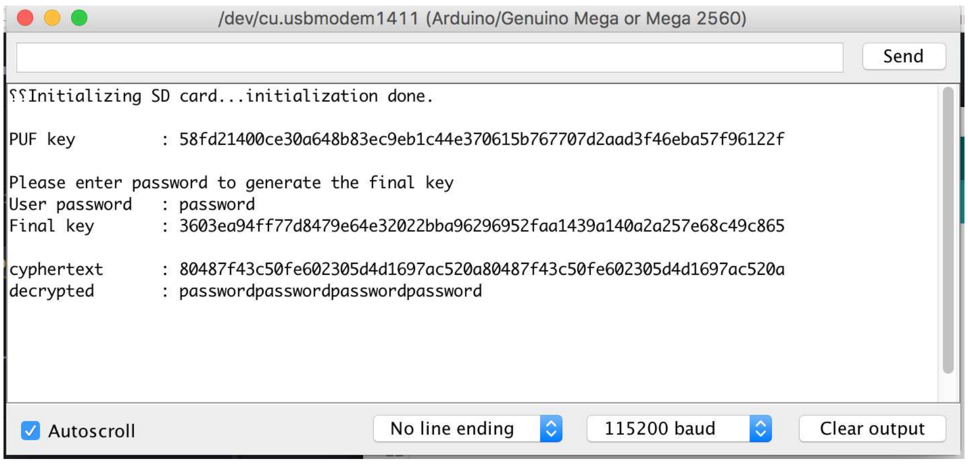
```
⸮⸮Initializing SD card...initialization done.

PUF key         : 58fd21400ce30a648b83ec9eb1c44e370615b767707d2aad3f46eba57f96122f

Please enter password to generate the final key
User password   : password
Final key       : 3603ea94ff77d8479e64e32022bba96296952faa1439a140a2a257e68c49c865

Please enter user's key (256-bit) to be encrypted
plaintext       : passwordpasswordpasswordpassword
encrypted       : 80487f43c50fe602305d4d1697ac520a80487f43c50fe602305d4d1697ac520a
```

Figure A.1: Screenshot of the Bitcoin key storing experiment during encryption stage using SRAM Cypress CY62256NLL ’A’. User’s password is ’password’ and the Bitcoin key (user’s key) is ’passwordpasswordpasswordpassword’.



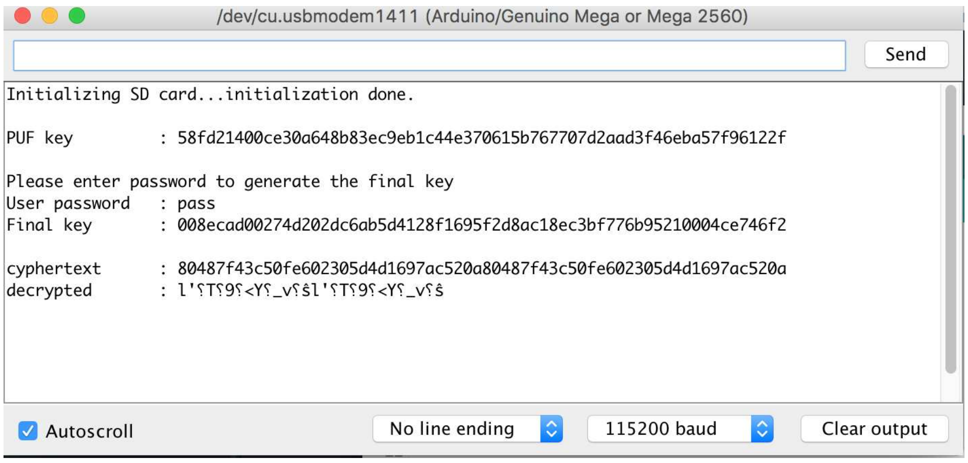
```
⸮⸮Initializing SD card...initialization done.

PUF key         : 58fd21400ce30a648b83ec9eb1c44e370615b767707d2aad3f46eba57f96122f

Please enter password to generate the final key
User password   : password
Final key       : 3603ea94ff77d8479e64e32022bba96296952faa1439a140a2a257e68c49c865

cyphertext      : 80487f43c50fe602305d4d1697ac520a80487f43c50fe602305d4d1697ac520a
decrypted       : passwordpasswordpasswordpassword
```

Figure A.2: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’passwordpasswordpasswordpassword’ is previously secured by using SRAM Cypress CY62256NLL ’A’ and user’s password ’password’. The Bitcoin key can be reconstructed because user’s password is correct and the utilized SRAM is SRAM Cypress CY62256NLL ’A’.



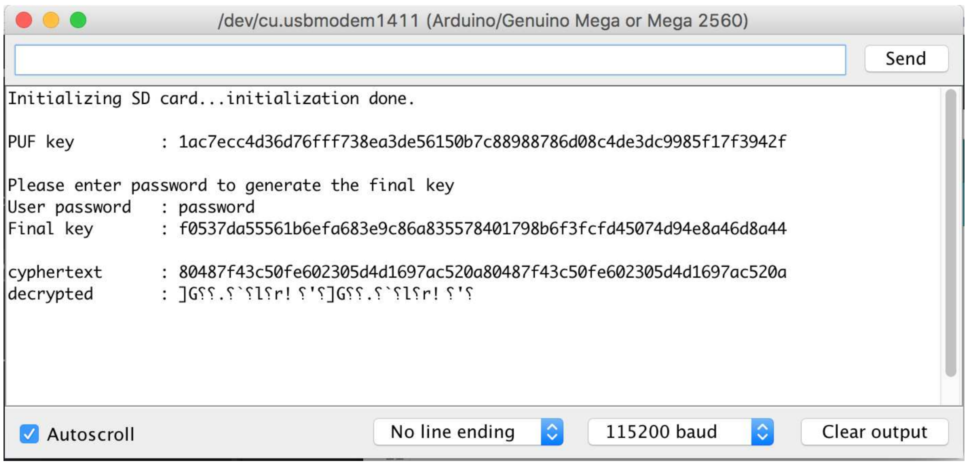
```
Initializing SD card...initialization done.

PUF key         : 58fd21400ce30a648b83ec9eb1c44e370615b767707d2aad3f46eba57f96122f

Please enter password to generate the final key
User password   : pass
Final key       : 008ecad00274d202dc6ab5d4128f1695f2d8ac18ec3bf776b95210004ce746f2

cyphertext      : 80487f43c50fe602305d4d1697ac520a80487f43c50fe602305d4d1697ac520a
decrypted       : l'⸮T⸮9⸮<Y⸮_v⸮ŝl'⸮T⸮9⸮<Y⸮_v⸮ŝ
```

Figure A.3: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’passwordpasswordpasswordpassword’ is previously secured by using SRAM Cypress CY62256NLL ’A’ and user’s password ’password’. Even though the utilized SRAM is the correct SRAM (SRAM Cypress CY62256NLL ’A’), the Bitcoin key cannot be reconstructed because user’s password is wrong.



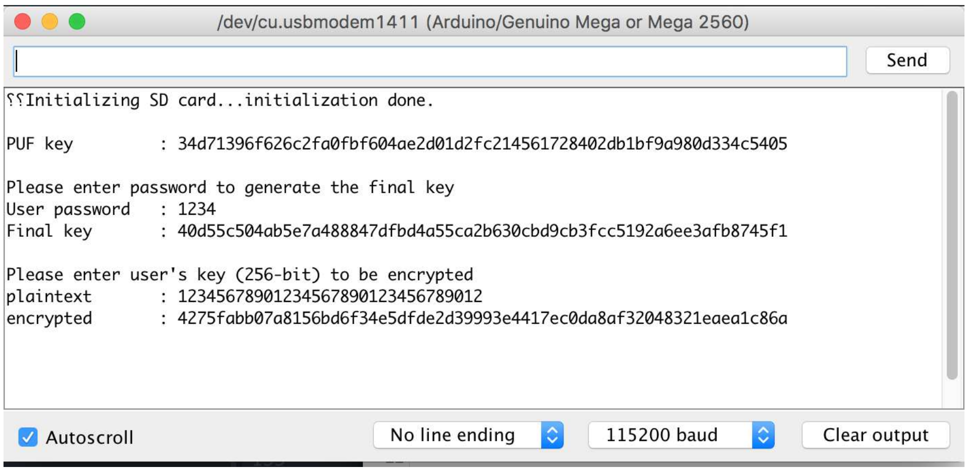
```
Initializing SD card...initialization done.

PUF key         : 1ac7ecc4d36d76fff738ea3de56150b7c88988786d08c4de3dc9985f17f3942f

Please enter password to generate the final key
User password   : password
Final key       : f0537da55561b6efa683e9c86a835578401798b6f3fcfd45074d94e8a46d8a44

cyphertext      : 80487f43c50fe602305d4d1697ac520a80487f43c50fe602305d4d1697ac520a
decrypted       : ]G⸮⸮.⸮`⸮l⸮r! ⸮'⸮]G⸮⸮.⸮`⸮l⸮r! ⸮'⸮
```

Figure A.4: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’1234567890123456’ is previously secured by using SRAM Cypress CY62256NLL ’A’ and user’s password ’password’. Even though user’s password is correct, the Bitcoin key cannot be reconstructed because the SRAM utilized for the decryption is SRAM Cypress CY62256NLL ’D’.



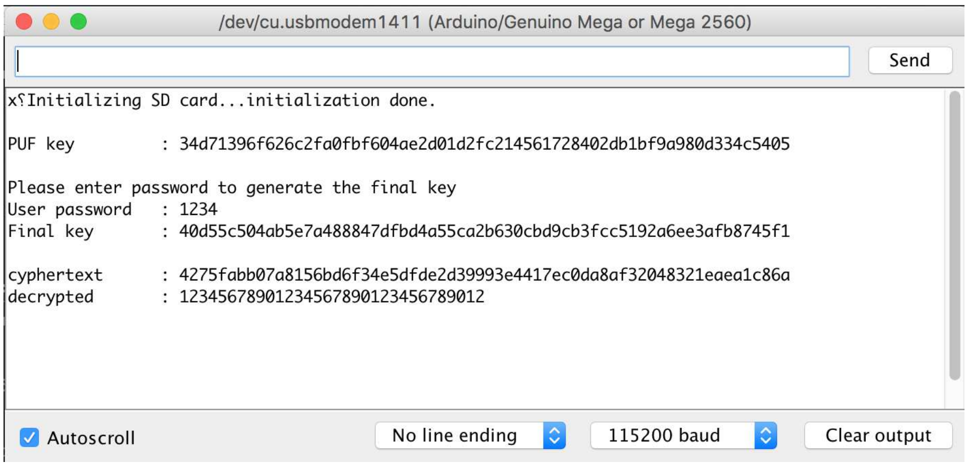
```
⸮⸮Initializing SD card...initialization done.

PUF key         : 34d71396f626c2fa0fbf604ae2d01d2fc214561728402db1bf9a980d334c5405

Please enter password to generate the final key
User password   : 1234
Final key       : 40d55c504ab5e7a488847dfbd4a55ca2b630cbd9cb3fcc5192a6ee3afb8745f1

Please enter user's key (256-bit) to be encrypted
plaintext       : 12345678901234567890123456789012
encrypted       : 4275fabb07a8156bd6f34e5dfde2d39993e4417ec0da8af32048321eaea1c86a
```

Figure A.5: Screenshot of the Bitcoin key storing experiment during encryption stage using SRAM Cypress CY62256NLL ’B’. User’s password is ’1234’ and the Bitcoin key (user’s key) is ’12345678901234567890123456789012’.



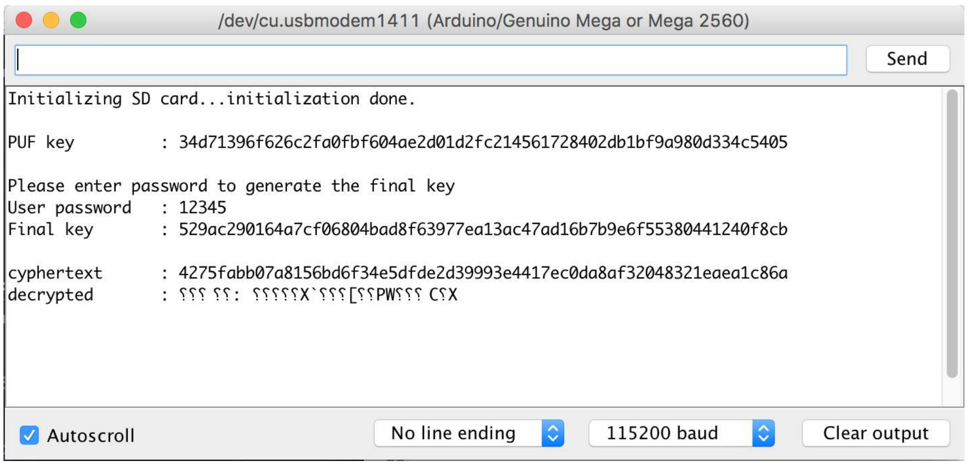
```
x؟Initializing SD card...initialization done.

PUF key          : 34d71396f626c2fa0fbf604ae2d01d2fc214561728402db1bf9a980d334c5405

Please enter password to generate the final key
User password    : 1234
Final key        : 40d55c504ab5e7a488847dfbd4a55ca2b630cbd9cb3fcc5192a6ee3afb8745f1

cyphertext       : 4275fabb07a8156bd6f34e5dfde2d39993e4417ec0da8af32048321eaea1c86a
decrypted        : 12345678901234567890123456789012
```

Figure A.6: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’12345678901234567890123456789012’ is previously secured by using SRAM Cypress CY62256NLL ’B’ and user’s password ’1234’. The Bitcoin key can be reconstructed because user’s password is correct and the utilized SRAM is SRAM Cypress CY62256NLL ’B’.



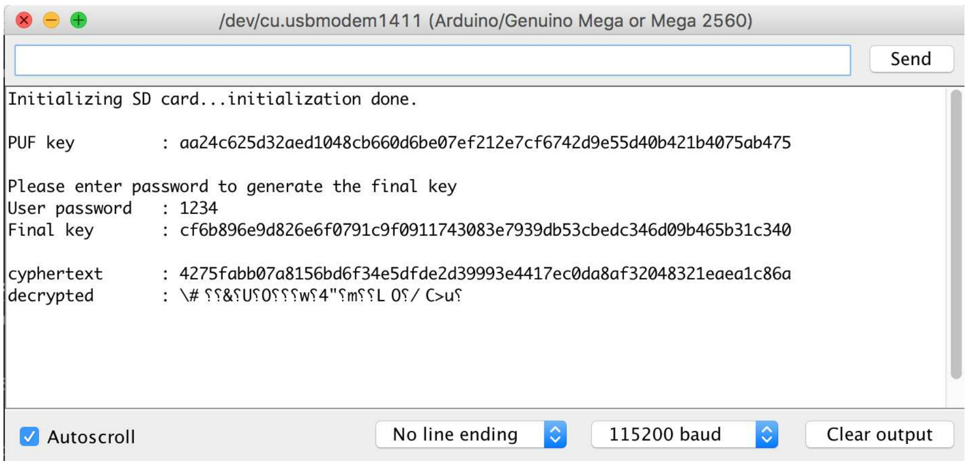
```
Initializing SD card...initialization done.

PUF key          : 34d71396f626c2fa0fbf604ae2d01d2fc214561728402db1bf9a980d334c5405

Please enter password to generate the final key
User password    : 12345
Final key        : 529ac290164a7cf06804bad8f63977ea13ac47ad16b7b9e6f55380441240f8cb

cyphertext       : 4275fabb07a8156bd6f34e5dfde2d39993e4417ec0da8af32048321eaea1c86a
decrypted        : ؟؟؟ ؟؟:  ؟؟؟؟؟X`؟؟؟[؟؟PW؟؟؟ C؟X
```

Figure A.7: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’12345678901234567890123456789012’ is previously secured by using SRAM Cypress CY62256NLL ’B’ and user’s password ’1234’. Even though the utilized SRAM is the correct SRAM (SRAM Cypress CY62256NLL ’B’), the Bitcoin key cannot be reconstructed because user’s password is wrong.



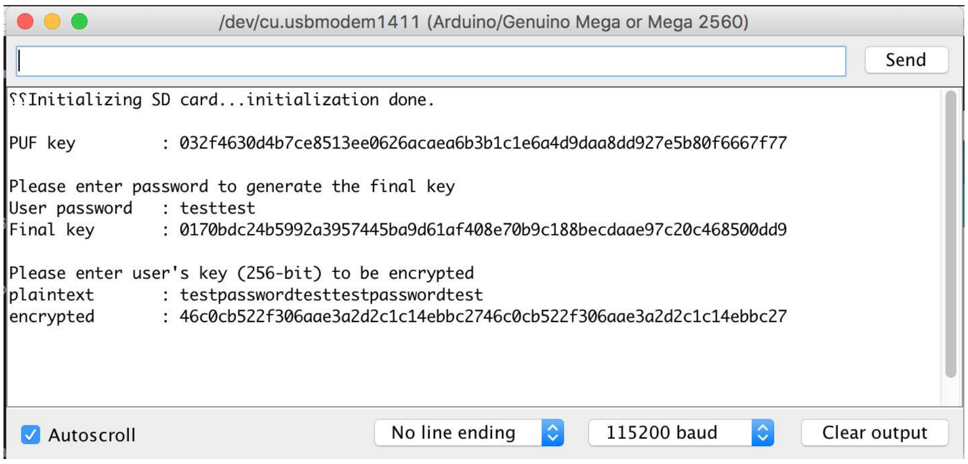
```
Initializing SD card...initialization done.

PUF key         : aa24c625d32aed1048cb660d6be07ef212e7cf6742d9e55d40b421b4075ab475

Please enter password to generate the final key
User password   : 1234
Final key       : cf6b896e9d826e6f0791c9f0911743083e7939db53cbedc346d09b465b31c340

cyphertext      : 4275fabb07a8156bd6f34e5dfde2d39993e4417ec0da8af32048321eaea1c86a
decrypted       : \# ؟؟&؟U؟O؟؟؟w؟4"؟m؟؟L O؟/ C>u؟
```

Figure A.8: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key '12345678901234567890123456789012' is previously secured by using SRAM Cypress CY62256NLL 'B' and user's password '1234'. Even though user's password is correct, the Bitcoin key cannot be reconstructed because the SRAM utilized for the decryption is SRAM Cypress CY62256NLL 'A'.



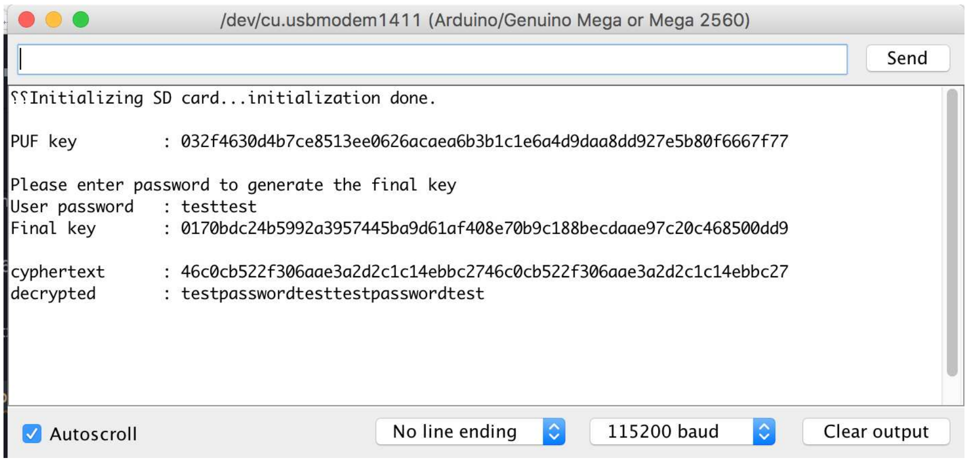
```
؟؟Initializing SD card...initialization done.

PUF key         : 032f4630d4b7ce8513ee0626acaea6b3b1c1e6a4d9daa8dd927e5b80f6667f77

Please enter password to generate the final key
User password   : testtest
Final key       : 0170bdc24b5992a3957445ba9d61af408e70b9c188becdaae97c20c468500dd9

Please enter user's key (256-bit) to be encrypted
plaintext       : testpasswordtesttestpasswordtest
encrypted       : 46c0cb522f306aae3a2d2c1c14ebbc2746c0cb522f306aae3a2d2c1c14ebbc27
```

Figure A.9: Screenshot of the Bitcoin key storing experiment during encryption stage using SRAM Cypress CY62256NLL 'C'. User's password is 'testtest' and the Bitcoin key (user's key) is 'testpasswordtesttestpasswordtest'.



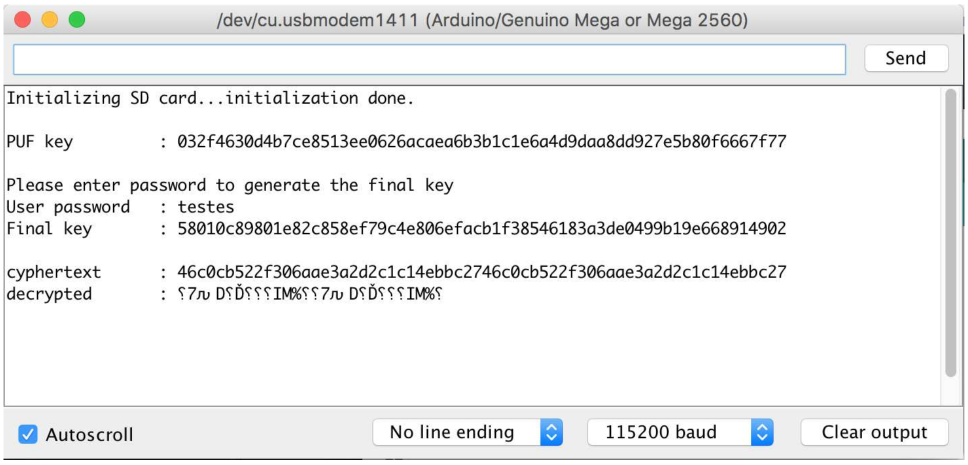
```
ʕʕInitializing SD card...initialization done.

PUF key         : 032f4630d4b7ce8513ee0626acaea6b3b1c1e6a4d9daa8dd927e5b80f6667f77

Please enter password to generate the final key
User password   : testtest
Final key       : 0170bdc24b5992a3957445ba9d61af408e70b9c188becdaae97c20c468500dd9

cyphertext      : 46c0cb522f306aae3a2d2c1c14ebbc2746c0cb522f306aae3a2d2c1c14ebbc27
decrypted       : testpasswordtesttestpasswordtest
```

Figure A.10: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’testpasswordtesttestpasswordtest’ is previously secured by using SRAM Cypress CY62256NLL ’C’ and user’s password ’testtest’. The Bitcoin key can be reconstructed because user’s password is correct and the utilized SRAM is SRAM Cypress CY62256NLL ’C’.



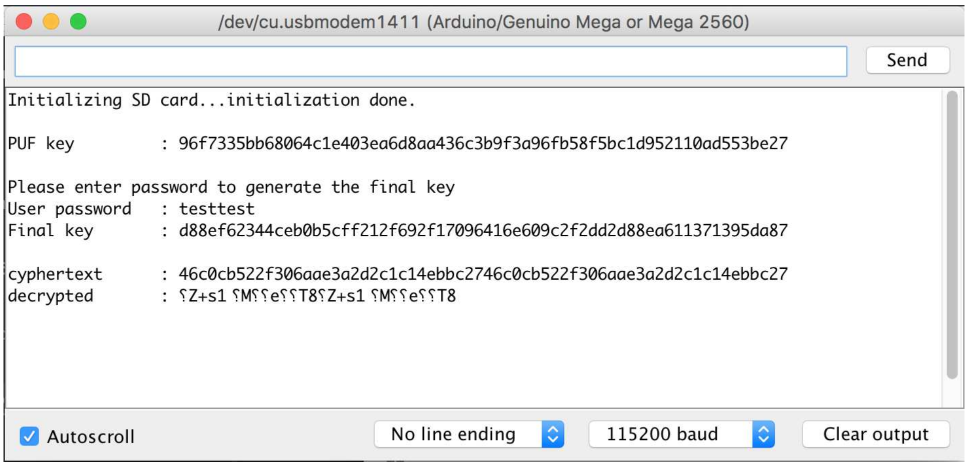
```
Initializing SD card...initialization done.

PUF key         : 032f4630d4b7ce8513ee0626acaea6b3b1c1e6a4d9daa8dd927e5b80f6667f77

Please enter password to generate the final key
User password   : testes
Final key       : 58010c89801e82c858ef79c4e806efacb1f38546183a3de0499b19e668914902

cyphertext      : 46c0cb522f306aae3a2d2c1c14ebbc2746c0cb522f306aae3a2d2c1c14ebbc27
decrypted       : ʕ7ກ DʕĎʕʕʕIM%ʕʕ7ກ DʕĎʕʕʕIM%ʕ
```

Figure A.11: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’testpasswordtesttestpasswordtest’ is previously secured by using SRAM Cypress CY62256NLL ’C’ and user’s password ’testtest’. Even though the utilized SRAM is the correct SRAM (SRAM Cypress CY62256NLL ’C’), the Bitcoin key cannot be reconstructed because user’s password is wrong.



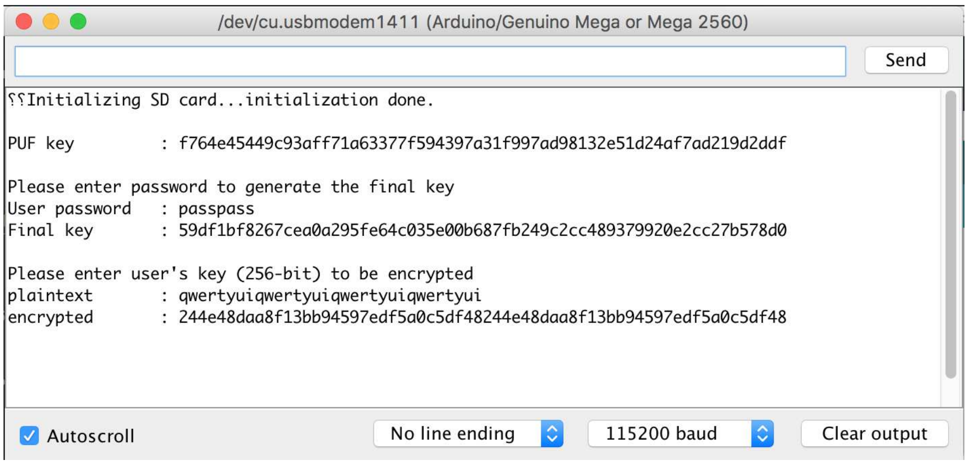
```
Initializing SD card...initialization done.

PUF key         : 96f7335bb68064c1e403ea6d8aa436c3b9f3a96fb58f5bc1d952110ad553be27

Please enter password to generate the final key
User password   : testtest
Final key       : d88ef62344ceb0b5cff212f692f17096416e609c2f2dd2d88ea611371395da87

cyphertext      : 46c0cb522f306aae3a2d2c1c14ebbc2746c0cb522f306aae3a2d2c1c14ebbc27
decrypted       : ʕZ+s1 ʕMʕʕeʕʕT8ʕZ+s1 ʕMʕʕeʕʕT8
```

Figure A.12: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’testpasswordtesttestpasswordtest’ is previously secured by using SRAM Cypress CY62256NLL ’C’ and user’s password ’testtest’. Even though user’s password is correct, the Bitcoin key cannot be reconstructed because the SRAM utilized for the decryption is SRAM Cypress CY62256NLL ’D’.



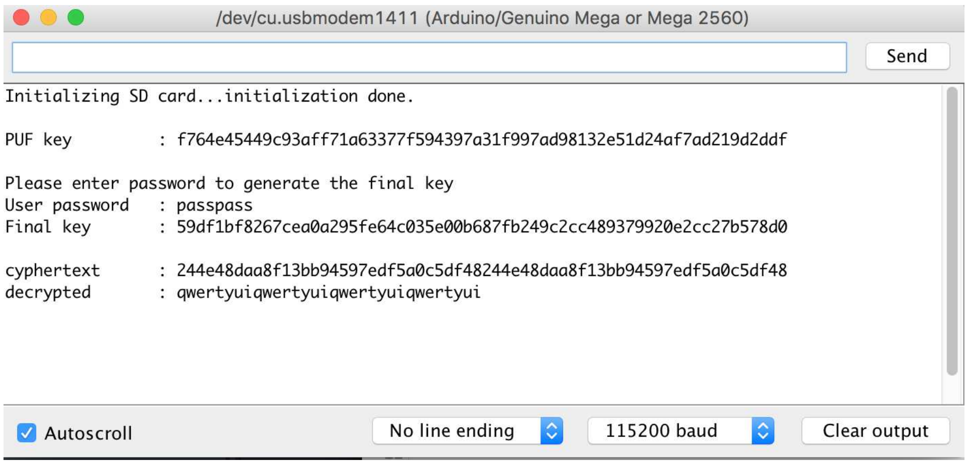
```
ʕʕInitializing SD card...initialization done.

PUF key         : f764e45449c93aff71a63377f594397a31f997ad98132e51d24af7ad219d2ddf

Please enter password to generate the final key
User password   : passpass
Final key       : 59df1bf8267cea0a295fe64c035e00b687fb249c2cc489379920e2cc27b578d0

Please enter user's key (256-bit) to be encrypted
plaintext       : qwertyuiqwertyuiqwertyuiqwertyui
encrypted       : 244e48daa8f13bb94597edf5a0c5df48244e48daa8f13bb94597edf5a0c5df48
```

Figure A.13: Screenshot of the Bitcoin key storing experiment during encryption stage using SRAM Cypress CY62256NLL ’D’. User’s password is ’passpass’ and the Bitcoin key (user’s key) is ’qwertyuiqwertyuiqwertyuiqwertyui’.



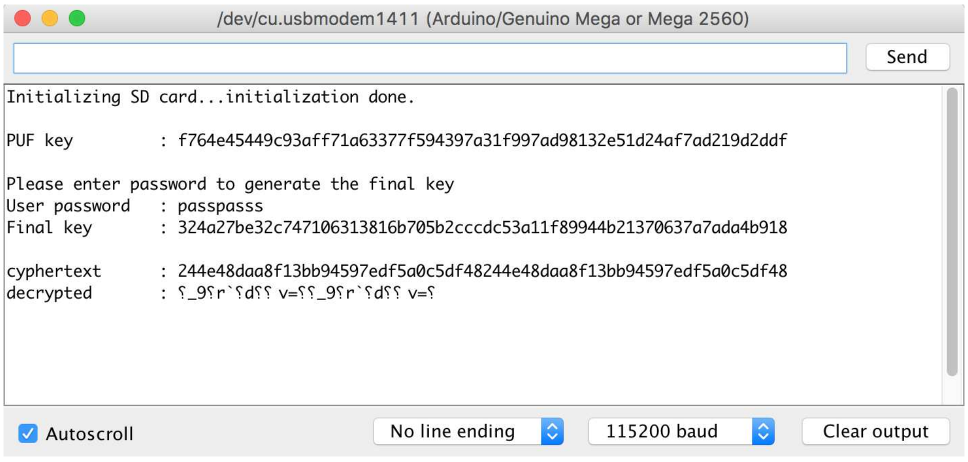
```
Initializing SD card...initialization done.

PUF key         : f764e45449c93aff71a63377f594397a31f997ad98132e51d24af7ad219d2ddf

Please enter password to generate the final key
User password   : passpass
Final key       : 59df1bf8267cea0a295fe64c035e00b687fb249c2cc489379920e2cc27b578d0

cyphertext      : 244e48daa8f13bb94597edf5a0c5df48244e48daa8f13bb94597edf5a0c5df48
decrypted       : qwertyuiqwertyuiqwertyuiqwertyui
```

Figure A.14: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’qwertyuiqwertyuiqwertyuiqwertyui’ is previously secured by using SRAM Cypress CY62256NLL ’D’ and user’s password ’passpass’. The Bitcoin key can be reconstructed because user’s password is correct and the utilized SRAM is SRAM Cypress CY62256NLL ’D’.



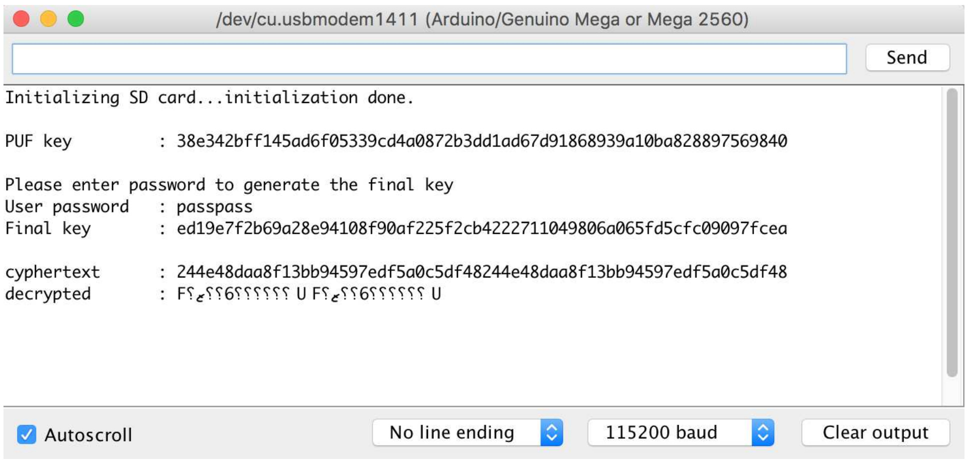
```
Initializing SD card...initialization done.

PUF key         : f764e45449c93aff71a63377f594397a31f997ad98132e51d24af7ad219d2ddf

Please enter password to generate the final key
User password   : passpasss
Final key       : 324a27be32c747106313816b705b2cccdc53a11f89944b21370637a7ada4b918

cyphertext      : 244e48daa8f13bb94597edf5a0c5df48244e48daa8f13bb94597edf5a0c5df48
decrypted       : ⸮_9⸮r`⸮d⸮⸮ v=⸮⸮_9⸮r`⸮d⸮⸮ v=⸮
```

Figure A.15: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’qwertyuiqwertyuiqwertyuiqwertyui’ is previously secured by using SRAM Cypress CY62256NLL ’D’ and user’s password ’passpass’. Even though the utilized SRAM is the correct SRAM (SRAM Cypress CY62256NLL ’D’), the Bitcoin key cannot be reconstructed because user’s password is wrong.



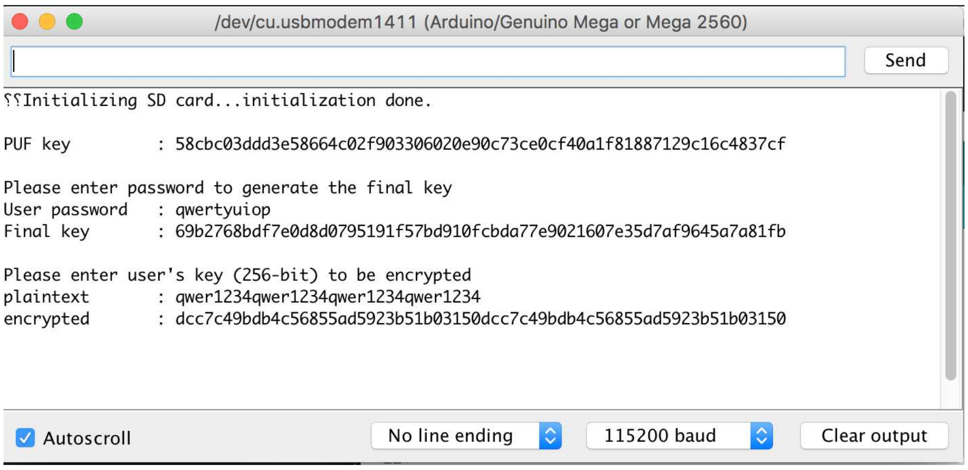
```
Initializing SD card...initialization done.

PUF key         : 38e342bff145ad6f05339cd4a0872b3dd1ad67d91868939a10ba828897569840

Please enter password to generate the final key
User password   : passpass
Final key       : ed19e7f2b69a28e94108f90af225f2cb4222711049806a065fd5cfc09097fcea

cyphertext      : 244e48daa8f13bb94597edf5a0c5df48244e48daa8f13bb94597edf5a0c5df48
decrypted       : F؟ع؟؟6؟؟؟؟؟؟ U F؟ع؟؟6؟؟؟؟؟؟ U
```

Figure A.16: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’qwertyuiqwertyuiqwertyuiqwertyui’ is previously secured by using SRAM Cypress CY62256NLL ’D’ and user’s password ’passpass’. Even though user’s password is correct, the Bitcoin key cannot be reconstructed because the SRAM utilized for the decryption is SRAM Cypress CY62256NLL ’B’.



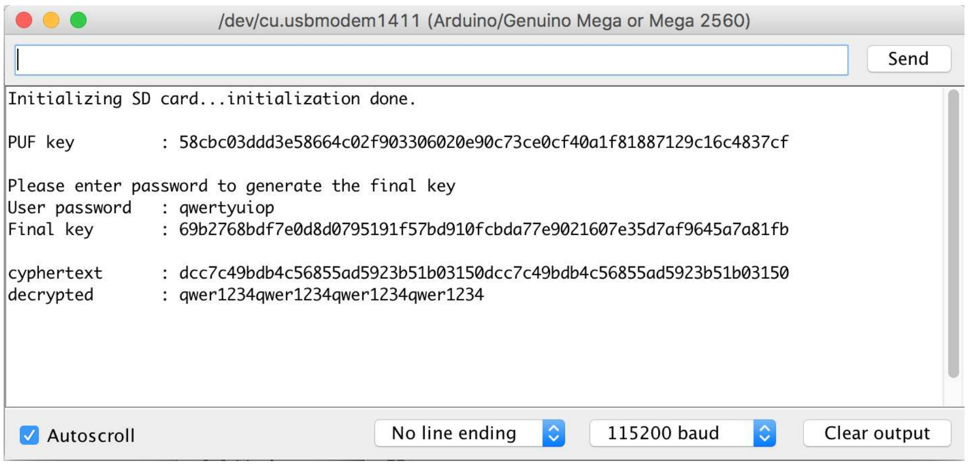
```
؟؟Initializing SD card...initialization done.

PUF key         : 58cbc03ddd3e58664c02f903306020e90c73ce0cf40a1f81887129c16c4837cf

Please enter password to generate the final key
User password   : qwertyuiop
Final key       : 69b2768bdf7e0d8d0795191f57bd910fcbda77e9021607e35d7af9645a7a81fb

Please enter user's key (256-bit) to be encrypted
plaintext       : qwer1234qwer1234qwer1234qwer1234
encrypted       : dcc7c49bdb4c56855ad5923b51b03150dcc7c49bdb4c56855ad5923b51b03150
```

Figure A.17: Screenshot of the Bitcoin key storing experiment during encryption stage using SRAM Cypress CY62256NLL ’E’. User’s password is ’qwertyuiop’ and the Bitcoin key (user’s key) is ’qwer1234qwer1234qwer1234qwer1234’.



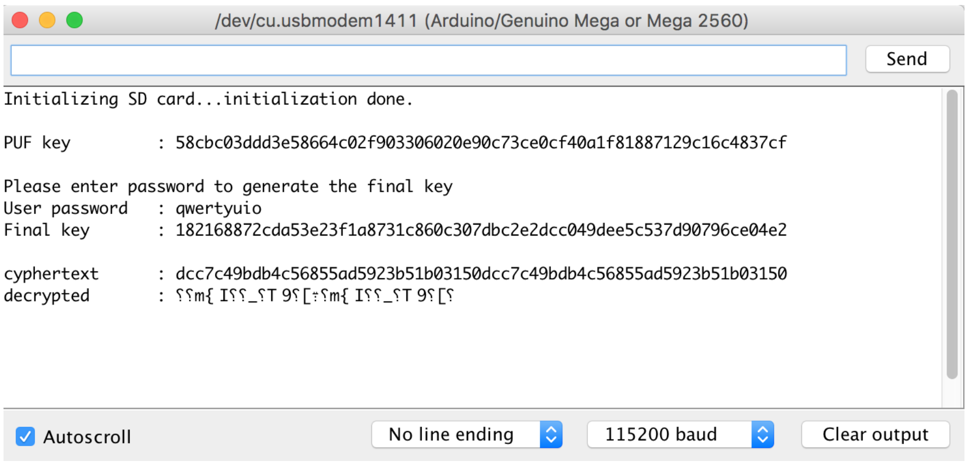
```
Initializing SD card...initialization done.

PUF key         : 58cbc03ddd3e58664c02f903306020e90c73ce0cf40a1f81887129c16c4837cf

Please enter password to generate the final key
User password   : qwertyuiop
Final key       : 69b2768bdf7e0d8d0795191f57bd910fcbda77e9021607e35d7af9645a7a81fb

cyphertext      : dcc7c49bdb4c56855ad5923b51b03150dcc7c49bdb4c56855ad5923b51b03150
decrypted       : qwer1234qwer1234qwer1234qwer1234
```

Figure A.18: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key ’qwer1234qwer1234qwer1234qwer1234’ is previously secured by using SRAM Cypress CY62256NLL ’E’ and user’s password ’qwertyuiop’. The Bitcoin key can be reconstructed because user’s password is correct and the utilized SRAM is SRAM Cypress CY62256NLL ’E’.



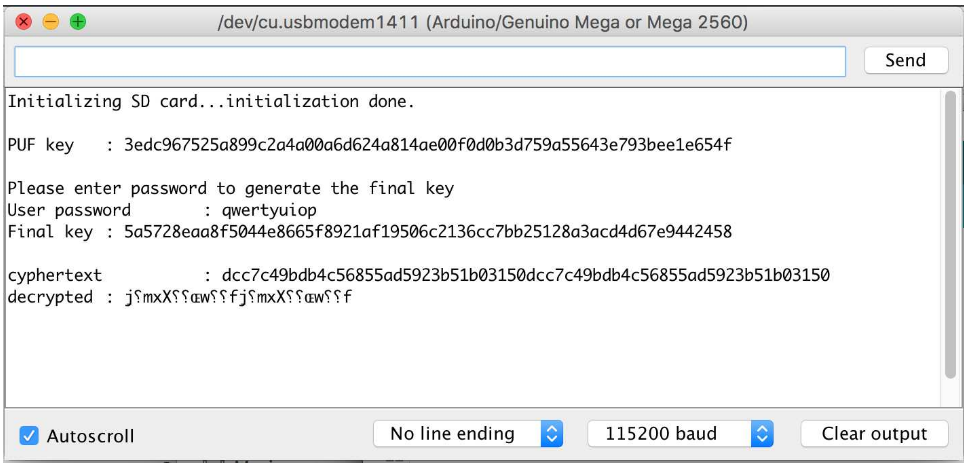
```
Initializing SD card...initialization done.

PUF key         : 58cbc03ddd3e58664c02f903306020e90c73ce0cf40a1f81887129c16c4837cf

Please enter password to generate the final key
User password   : qwertyuio
Final key       : 182168872cda53e23f1a8731c860c307dbc2e2dcc049dee5c537d90796ce04e2

cyphertext      : dcc7c49bdb4c56855ad5923b51b03150dcc7c49bdb4c56855ad5923b51b03150
decrypted       : ʕʕm{ Iʕʕ_ʕT 9ʕ[÷ʕm{ Iʕʕ_ʕT 9ʕ[ʕ
```

Figure A.19: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin ’qwer1234qwer1234qwer1234qwer1234’ is previously secured by using SRAM Cypress CY62256NLL ’E’ and user’s password ’qwertyuiop’. Even though the utilized SRAM is the correct SRAM (SRAM Cypress CY62256NLL ’E’), the Bitcoin key ’testpasswordtest’ cannot be reconstructed because user’s password is incorrect .



```
Initializing SD card...initialization done.

PUF key    : 3edc967525a899c2a4a00a6d624a814ae00f0d0b3d759a55643e793bee1e654f

Please enter password to generate the final key
User password        : qwertyuiop
Final key  : 5a5728eaa8f5044e8665f8921af19506c2136cc7bb25128a3acd4d67e9442458

cyphertext           : dcc7c49bdb4c56855ad5923b51b03150dcc7c49bdb4c56855ad5923b51b03150
decrypted  : jʕmxXʕʕœwʕʕfjʕmxXʕʕœwʕʕf
```

Figure A.20: Screenshot of the Bitcoin key storing experiment during decryption stage. The Bitcoin key 'qwer1234qwer1234qwer1234qwer1234' is previously secured by using SRAM Cypress CY62256NLL 'E' and user's password 'qwertyuiop'. Even though user's password is correct, the Bitcoin key cannot be reconstructed because the SRAM utilized for the decryption is SRAM Cypress CY62256NLL 'C'.