HAL open science

PUF based Secure Computing for Constraint Cyber Physical Object

Amir Alipour

To cite this version:

Amir Alipour. PUF based Secure Computing for Constraint Cyber Physical Object. Micro and nanotechnologies/Microelectronics. Université Grenoble Alpes [2020-..], 2022. English. NNT : 2022GRALT097 . tel-04025751

HAL Id: tel-04025751

https://theses.hal.science/tel-04025751v1

Submitted on 13 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : EEATS - Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)
Spécialité : Nano électronique et Nano technologies
Unité de recherche : Laboratoire de conception et d'intégration des systèmes

Utilisation des fonction physiques non clonages pour la Securisation des systemes embarques

PUF based Secure Computing for Constraint Cyber Physical Object

Présentée par :

Amir ALIPOUR

Direction de thèse :

David HELY MAITRE DE CONFERENCES, Université Grenoble Alpes	Directeur de thèse
Giorgio DI NATALE DIRECTEUR DE RECHERCHE, Université Grenoble Alpes	Co-encadrant de thèse
Vincent BEROULLE PROFESSEUR DES UNIVERSITES, GRENOBLE INP	Co-encadrant de thèse

Rapporteurs :

Alberto BOSIO
PROFESSEUR DES UNIVERSITES, ECOLE CENTRALE LYON
Jean-Luc DANGER
PROFESSEUR DES UNIVERSITES, TELECOM PARIS

Thèse soutenue publiquement le **8 décembre 2022**, devant le jury composé de :

Alberto BOSIO PROFESSEUR DES UNIVERSITES, ECOLE CENTRALE LYON	Rapporteur
Jean-Luc DANGER PROFESSEUR DES UNIVERSITES, TELECOM PARIS	Rapporteur
Fatemeh AFGHAH PROFESSEUR ASSOCIE, Clemson University South Carolina	Examinatrice
Laurent FESQUET MAITRE DE CONFERENCES HDR, GRENOBLE INP	Examinateur
Lilian BOSSUET PROFESSEUR DES UNIVERSITES, UNIVERSITE DE SAINT-ETIENNE - JEAN MONNET	Président
David HELY MAITRE DE CONFERENCES HDR, GRENOBLE INP	Directeur de thèse

Invités :

Giorgio Di Natale
DIRECTEUR DE RECHERCHE, CNRS
Vincent Beroulle
PROFESSEUR, Grenoble INP

ACKNOWLEDGEMENT

I would like to thank my family first as the forerunners of support in my life. Vida Hagh-Parast, my mother, whose beautiful support pushed me to come so far in life, and whose vision for a brighter future for me and my brother has always been the number one encouragement, Mohammadreza Ali-pour, my father, who has been the most reliable support in my life, and the toughest and the most caring teacher, and Navid Ali-pour, my brother, who always taught me I could do better by him simply being the better version of me, and for being there for me whenever I needed him. I thank them sincerely for being there for me for the past three years of my PhD studies, and dedicate this dissertation to them to reciprocate their support.

I deeply thank my thesis director and co-supervisors, Dr. David Hely, Prof. Vincent Beroulle, and Dr. Giorgio Di Natale. Their support has been immense and critical in pulling this work out into a bright sight. This past three years I learnt many things from them that surely makes a huge impact in my future career. A sincere thanks also to Dr. Fatemeh Afghah, whose support for two years of my PhD studies is worth mentioning. I worked with Dr. Afghah remotely and on site in Clemson University, and at all times I received lots of support and critical supervision in our collaborative work.

I would finally thank a group of amazing friends and colleagues that have been around me in the past three years and have supported me. I thank first of all Dr. Zahra (Elnaz) Kazemi, an amazing friend and the forerunner in support among friends. And I thank Mr. Ashkan Azarfar, Mr. Mahdi Talebi, Mr. Amin Seyed Haeri, Mr. Ali Owfi, Mr. Mahdi Shayan, Mme. Leila Shadmani, Dr. Amir-Pasha Mir Baha, Dr. Raphael Tavares De Alencar, Mr. Raymundo De Amorim junor, Mr. Florian Requena, Mr. Omid Bazangani, Dr. Nicolas Barbot, Dr. Dahmane Allane, Dr. Zeshan Ali, Mme. Caroline Palisse, Mme. Marie Vergriete, Mr. Arthur Desuert, and Mr. Arthur Baudet.

A special thanks to Université Grenoble Alpes for giving me this opportunity and supporting me to pursue my PhD studies with them. I would like to mention that my PhD studies has been supported financially by the French National Research Agency in the framework of the "Investissements d'avenir" program (ANR-15-IDEX-02).

TABLE OF CONTENTS

1 Introduction **9**

Introduction **9**

2 State of the Art **15**

2.1 Introduction 15

2.2 Description of PUF & It's Implementation 16

Description of PUF & It's Implementation 16

2.3 PUF numerical characteristic and parameters 16

PUF numerical characteristic and parameters 16

2.4 PUF Structure Variations 20

PUF Structure Variations 20

2.4.1 Arbiter PUF and its sub-variants 20

2.5 Conventional PUF Protocols 25

Conventional PUF Protocols 25

2.5.1 Fundamental steps of using PUF 26

Existing Commercial PUFs 29

2.6 Challenges, Opportunities, and the Motivations 30

Challenges, Opportunities, and the Motivations of this Work 30

2.7 Machine Learning and PUF Modeling 32

2.8 What Predictive Model to Use 34

2.8.1 Logistic Regression 35

2.8.2 Support Vector Machine 35

2.8.3 Decision Tree and Random Forest 36

2.8.4 Multi-layer Perceptron 37

2.8.5 Convolutional Neural Network 39

2.8.6 Discusion 39

2.9 Using PUF Simulation to Evaluate Real PUF 41

3 PUF Modeling and CRP Data-base Management 43
3.1 Introduction 43
3.2 Strong PUF Enrollment using Machine Learning: A Methodical Approach 45
3.2.1 Initialization Phase 47
3.2.2 Optimization Phase 48
3.2.3 Evaluation Phase 48
3.2.4 Methodology 49
3.2.5 Evaluation Work 51
3.2.6 Evaluation Observations 53
3.2.7 Applicability in the Related Works 57
3.3 Transfer Learning: A Modeling Technique to Reduce Training Data . . . 60
3.3.1 How Transfer Learning Works 61
3.3.2 Proposed Method 61
3.3.3 Experimental Setup 64
3.3.4 Experimental Results 65
3.4 Subspace Modeling: A Technique to Tackle Requirement for Large CRP data for training 69
3.4.1 Sub-Space Modeling Method 70
3.4.2 Experimental Setup 71
3.4.3 Experimental Results 73
3.5 Conclusion 78

4 Implementation of PUF-based Protocols 81
4.1 Introduction 81
4.2 Introducing Helper Data Masking: An Improvement for FE-based Key Exchange Protocols 83
4.2.1 Helper Data Manipulation Attack 84
4.2.2 Helper Data Masking ith Variable Positioning 86
4.2.3 Mask Vector Generation 86
4.2.4 Masking with Variable Positioning 89
4.2.5 Vulnerability of Masking against HDM Attack 90
4.2.6 Detection of Weak Mask Vector 92
4.3 Theoretical Analysis of masking with variable positioning 94
4.3.1 Evaluation on various Coding and Decoding methods 98

4.4 Experimental Setup 99
4.4.1 Description of SRAM PUF device 99
4.4.2 PUF Data Description 100
4.5 Experimental results 101
4.6 Related Works and Comparison 103
4.7 Designing a Novel Secure Key Exchange Technique using ML and Strong PUF 107
4.7.1 Proposed Method 108
4.7.2 Related Works 111
4.7.3 Evaluation of Reliablity 112
4.7.4 Security Analysis 114
4.8 Utilization of Simulatable PUFs and its Constituents 118
4.9 Conclusion 119

5 Reviewing Challenges and Solutions, and Disscusion on Applications 123
5.1 Introduction 123
5.2 Open Challenges in Transfer Learning 124
5.3 Open Challenges in Subspace Modeling 126
5.4 Mixing Modeling Solutions to Yield more Optimal Results 128
5.5 Anticipated Improvements in the Secure key Exchange Protocol 129
5.6 More Opportunities with Coupling ML & PUF 130
5.7 Range of Applications using PUF 133
5.8 The Future of the RFE-like protocl and our Helper Data Masking Countermeasure 139
5.9 Conclusion 141

Conclusion 143

Bibliography 147

1 Introduction

Physically Unclonable Function (PUF) is known as a hardware security primitive [1]. On a silicon chip, PUF can be seen as a component founded on the concept of process variation to generate chip-specific values. This component as a function receives a challenge on the input and projects the response to the challenge on the output, and the challenge-response pair (CRP), singular or plural, will represent as the unique values. While structurally the same, PUFs on different chip instances generate values that are different and unique to the chip instance. PUF has been studied for more than two decades, and since then, many variations of it has been proposed [1], [2]. Numerous protocols have been designed also using PUF for digital device authentication and encryption key generation [3]. Up to now, there exists a very large family of hardware implementations of PUF. Trivially, they all fall into two major categories: Strong PUF; referring to any variant which can generate non-enumerable CRPs, and Weak PUF; referring to any variant which generates very few or often only one device specific unique value.

The mandatory requirement for correctly using a PUF-based system is to have a secure server, which is in charge of storing the secret information extracted from the PUF during the enrollment. In mission mode, the secure server is then involved in the establishment of secure protocols requiring the use of the PUF secret information. These protocols will, for instance, guarantee the authenticity of the hardware device, or will allow confidentiality through cryptographic primitives using the PUF value as master key. Figure **??** shows the conventional establishment of the enrollment mode and mission mode of a PUF-based eco-system.

While several techniques have been proposed in literature for the integration of PUF-

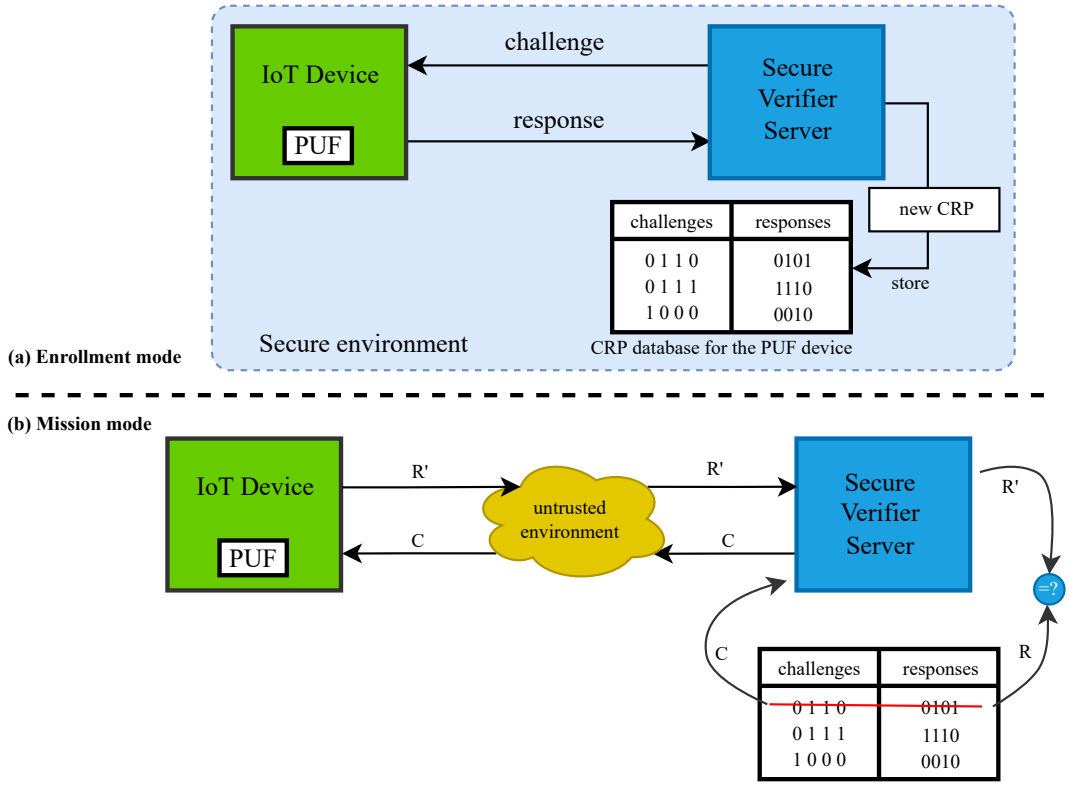


Figure 1.1: Illustration showing the traditional establishment of the enrollment and mission mode of a PUF-based eco-system.

based protocols to enhance authentication and communication protocols, the practical and industrial issues related to the enrollment management are seldom addressed. Few works such as the one discussed in [4] by Khalfaoui et al, reviews the possibility of employing machine learning for PUF modeling. In particular, both the longevity of the enrollment time, and the size of the enrollment data are still open problems. On one hand, we could imagine the demand of storing all the CRPs of a PUF. This is practically impossible since the CRPs of strong PUF is non enumerable. A simple example in this regard is to consider a PUF with a 128-bit challenge input. Such PUF can in turn generate 2^{128} responses. Even considering to save the responses only will not be possible due to the very large number of possible responses. On the other hand, considering to capture a very large number of CRPs per device can take a lot of time during the test phase where we mainly imagine to perform the CRP-readout. Surely the industrial constraints will not allow spending very long time per chip to capture a large dataset of CRPs.

To mitigate these issues with enrolling strong PUF, we can consider using machine learing with the goal to provide a model of PUF for enrollment and replace the CRP

database with it. To this day, several works have been conducted and proved successful in modeling PUF with machine learning, especially the deep learning techniques. However, at first sight, these modeling techniques have been introduced as potential tools for attackers that try to create a clone of a PUF instance [5]. For instance, it has been proved that a simple Multi-Layer Perceptron model can be trained to predict the response of several architecture of arbiter PUFs, by being trained with just a very small set of CRPs of the PUF-enabled device [6]. For this reason, many variations and countermeasures are proposed for every family of PUF architectures (such as double k-XOR arbiter PUF [7]) with the goal of avoiding model-building attacks using machine learning and deep learning techniques.

However, the potential of using modeling techniques in favor of PUF enrollment to save storage and time is promising. Indeed, replacing the PUF secret database by a trained model present several advantages. It would make practical (in terms of enrollment time and database size) security applications (e.g., protocols, authentication mechanisms) requiring a large amount of PUF secret data. The enrollment needs indeed to focus only on a subset of the data and then the trained model should be capable of generating any data that can provide the given PUF. Moreover, the PUF model could also be enhanced by additional variables and parameters related to environmental conditions or the aging of the integrated circuit. These additional properties should make possible the self-adjustment of the PUF responses during the mission mode in order to increase the reliability of the protocols and algorithms using the PUF. Figure 1.2 depicts the general schematic of what an eco-system including PUF and it's predictive equivalent model on the secure server looks like.

So we can see this as one possible solution: To rely on modeling techniques to generate a predictive model of the PUF instance as a reference on the secure server, which would replace the traditionally stored set of CRPs. We first introduced it in [8]. In this solution, the enrollment procedure would require a subset of the possible CRPs to perform machine learning on and provide a model of PUF. This subset would be smaller than the one required in the traditional approach for enrollment. Using this subset of CRPs and a suitable machine learning modeling solution, the model of the PUF is built and it is then used to predict the entire possible CRPs of the PUF instance.

To realize this potential, a re-introduction of PUF modeling to facilitate PUF utilization is required. In doing so, there are two major points to uphold:

- **Point 1:** We want to evaluate the feasibility of providing and storing the predictive

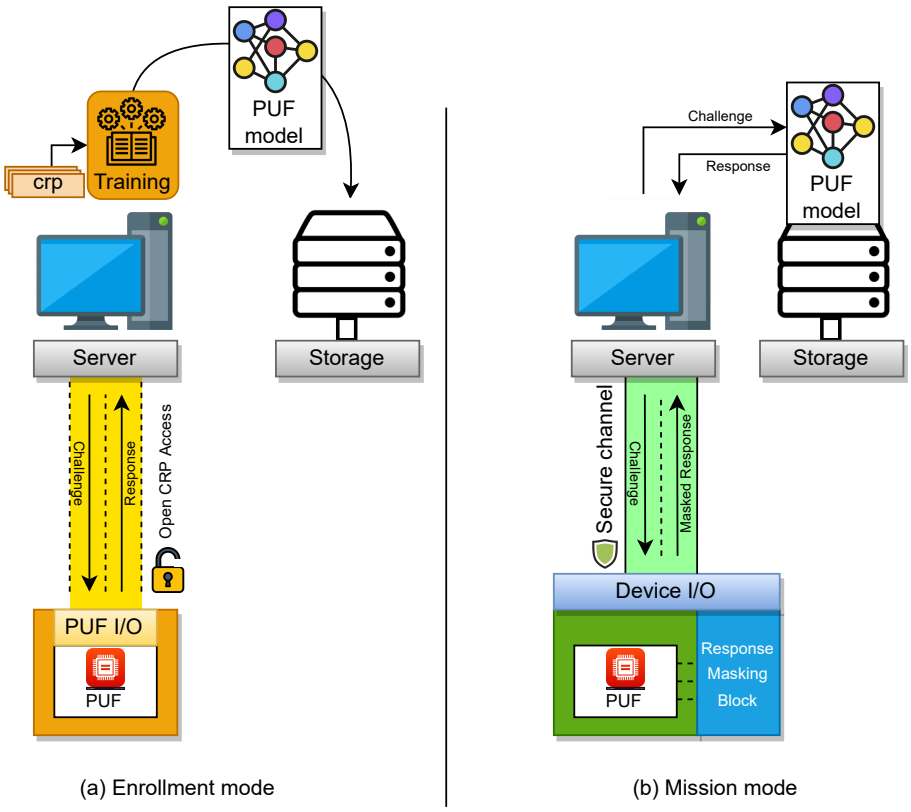


Figure 1.2: Illustration showing (a) the general schematic of enrolling PUF with Machine Learning and (b) utilizing the ML model to communicated with the PUF.

model of the PUF. It is critically important to ensure that the creation of a predictive model of the PUF is doable and imposes minimal cost in industrial settings.

- **Point 2:** We want to assure that the PUF and its equivalent predictive model are used in a protocol which is secure in a manner that is preventing modeling attacks taking the same advantage as it is given to the users in providing the model of PUF for beneficial use-case.

prior to performing machine learning, we need to know what modeling solution we should employ. There is a vast spectrum of possible machine learning solutions to choose from to model PUF with. In Chapter 1 we present a range of possible modeling techniques, and identify those that fit best the modeling of strong PUF. Commonly Neural Network models are potential to handle great deal of noise and complex data models, and they can achieve optimal behavior with acceptable amount of data, if properly trained. Their use cases are for many different purposes mainly including classification and pattern recognition. Their use case in the field of PUF modeling can greatly affect

the storage and time of enrollment compared to those in the traditional methods. We commonly use Neural Networks in this research work. However, for simpler modeling subjects, we employ simpler modeling methods like logistic regression.

To cost efficiently model PUF, we need to look at several parameters during the enrollment. On the PUF side, we look at the number of CRPs to collect from the PUF during the CRP-readout from the device under test (DUT). On the secure server side, we look at parameters such as the model's prediction accuracy, total time of training, and the predictive model size. In Chapter 2, we introduce these as cost parameters and then explain that there can be control parameters such as validation and test accuracy, prediction loss during training, and number of iterations during training, that are encapsulated into a method to control the machine learning to yield the desired model with minimum cost. We put more stress on the cost of training and how to reduce it even further. Later on in the same chapter, we introduce some optimization techniques for PUF modeling with the aim to reduce the overall cost training.

Since during enrollment the trusted party is able to model the PUF, it natural as well to induce that the PUF is vulnerable to modeling attack. However, we should notice that this vulnerability at first place is due to the exposure of the CRP or an offset of the CRP or PUF response to the public. Therefore, we can consider two possibilities: 1) to improve an existing protocol in a way to obfuscate or mask the publicly exposed PUF data, 2) to develop a new protocol which does not include exposure of any PUF related data. In Chapter 3, we introduce 2 of our contributions in this regard, one which is a masking mechanism on a publicly available code-offset of PUF response, and the other is a new protocol for authentication and key exchange which requires exchanging only user generated data to symmetrically provide secret values on communicating parties. We show further in that chapter that designing a new protocol as explained can be possible thanks to employing a model of the PUF on the secure server.

Up to this point, the end to end spectrum of various contributions we deliver in this work, and why we intend to deliver them has been discussed. However, details on each are yet to be presented in Chapter 2 and 3. In Chapter 4 also, we elaborate on some of the miscellaneous ideas that we thought of surrounding the coupling of PUF and machine learning. We also briefly talk about the possibility of employing a simulation of a strong PUF on a weak PUF substrate, an exotic development to allow us to have a large CRP space, to then allow us to employ the ml-based PUF protocol we designed earlier in Chapter 3.

In the next chapter, we present the state of the art, where we talk about the fundamentals of PUF and machine learning. Details on PUF, PUF variants and their characteristic, important metrics regarding PUF, as well as details on machine learning methods, variants of ML structures and effectiveness of each model structure for modeling PUF are all discussed in the next chapter.

2
State of the Art

2.1 Introduction

This chapter brings together some of the baseline information regarding Physical Unclonable Function (PUF), how PUF works, what are the characteristics that an ideal PUF has and what metrics we use to identify an ideal PUF, applications and protocols that utilize PUF, and the known challenges that coop with PUF utilization that exist since the advent of the concept, alongside the new challenges that exist to this day, some of which are the incentives of this PhD work. The take from this chapter is to prepare the reader of the fundamental concepts, before going deeper into the utilization of PUF and the parametric definitions that elaborate in more details the concept of PUF itself. The explanation and elaboration in this chapter is broad and to the surface.

At the core of this work, we use machine learning to model PUF for beneficial use-cases. To use ML in modeling PUF, we first need to know how ML works and what ML model we should choose according to what application or task we are considering it. Since variations of ML model solutions are in abundant, and often in choosing or creating a novel model, there are several parameters involved, we need to first know what these models are and what parameters we can modify to reach to the potential solution structure before the training begins. We talk about these points as well in this chapter. At the end of this chapter, the readers should know the base-knowledge of PUF, what type of PUF we are majorly concerned with in this work, basic information on machine learning and what model we use predominantly to model what type of PUF and why the model is best suited for that PUF variant.

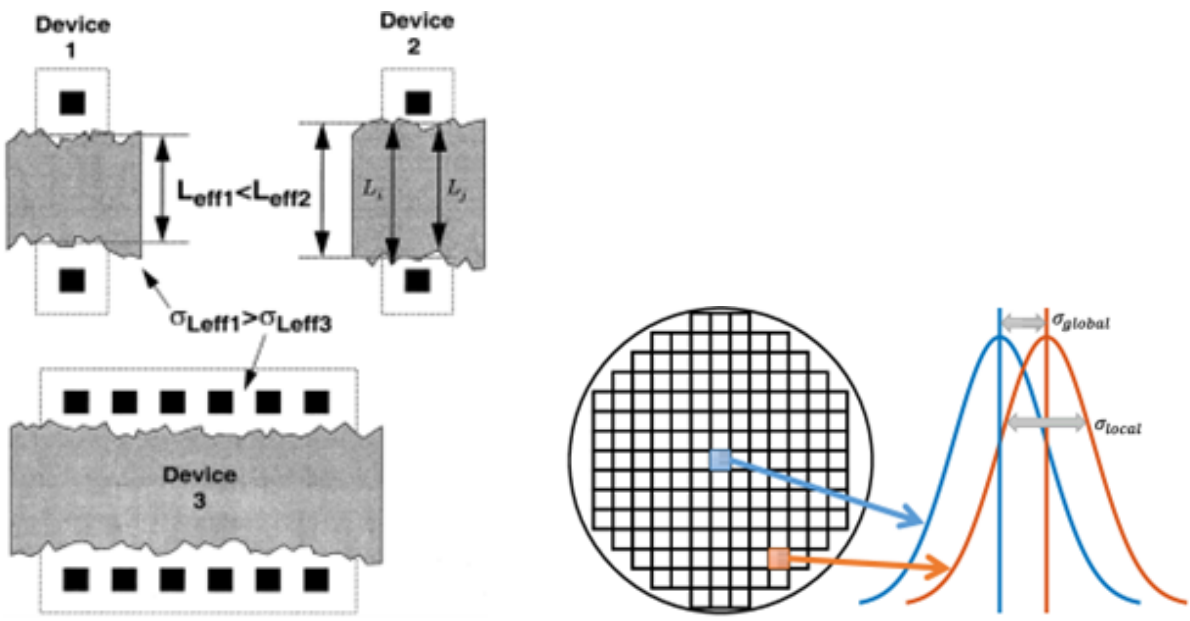
2.2 Description of PUF & It's Implementation

Silicon device manufacturing is where we can start explaining how PUF is created. Normally, during the manufacturing process, the digital devices are placed into the silicon substrate in form of a wafer. On a wafer also, the same blueprint of a chip or an ASIC processing unit is implemented on small dies. Each die implemented follows the same blueprint developed in the design process. This makes the expectation that every die is identical to another. While from a macroscopic vision, it is an expectation that is usually met, through a microscopic vision however, the case is different. A microscopic vision will show the smallest building blocks of each die, like a transistor. In such vision its often observed that two devices from the same position according to the blueprint, however each device belonging to a different die, will have minor differences such as the size of the transistor gate (see Figure.2.1b)). An observation like this is shown in Figure.2.1a. This is often referred to as micro process variation. Such difference will in turn lead to different signal propagation delay when we compare the performance of each device. Nonetheless this difference is often observed in the order of nanoseconds. This propagation indifference is often so minor that in the macroscopic vision, it is considered negligible. However, there is a possibility that specific designs that are aware of such indifference, aim to collect the micro process variations to represent a macroscopic effect [9]. Such implementation is the base-ground of how we can implement PUF.

PUF, as its name suggests, is a function. This function takes a challenge in the input and produces a response to that challenge on the output. The challenge and the response to it are corresponding, so they make a challenge response pair or CRP in abbreviation. Depending on the structure of the PUF, and the input challenge size, PUF can generate from a singular response and so a singular CRP in general, to billions of CRPs [1], [3]. PUFs that can generate large number of CRPs are called strong PUFs, and those with few CRPs are called weak PUF [2]. Later on, we explain what these variations are and how we can identify a good PUF characteristic.

2.3 PUF numerical characteristic and parameters

The main characteristic that is important to the PUF users is the CRP. While the micro-variation cultivation in the bare-metal level is as crucial as the desired functionality in terms of CRP, the user mostly pays attention to the CRP characteristics only, leaving the



(a) (b)

Figure 2.1: 1 a) Illustration showing the physical minor differentiation between two devices. b) showing the process variation between two manufactured chips on a wafer

bare-metal implementation to the attention of the designers.

From the perspective of a PUF user, an ideal CRP characteristic can be drawn. An ideal CRP characteristic should in turn be: 1) randomized: Meaning that the drawn CRPs in any order are not predictable or follow any pattern, 2) Unique: Meaning that the CRPs from one PUF device are unique only to that PUF device, 3) Well diffused: Meaning that in any sub-set of consecutively captured CRPs from a PUF, there is no dense co-location of 1s and 0s, or any small pattern comprising 0s and 1s, and 4) Reliable: Meaning that in multiple acquisitions of Responses for a given challenge or a set of challenges, the observed response in majority of the time (ideally all the time) is or are the same. These characteristic metrics can be numerically and logically measured. In the following we introduce the measurements for each of the metrics:

Randomness

Measuring the randomness of PUF was first introduced in [10]. According to the original description of Randomness in PUF CRP characteristic, the number of 0s and 1s are approximately (ideally the same) equal. This means that from any subset of CRPs collection from a PUF instance, the number of 0s and 1s are close equal to the half of the

total size of the set. We can measure the randomness as follows:

$$H = -\log_2 max(p, 1-p) \tag{2.1}$$

where p is represents the frequency of 1's appearing in a set of CRPs. p itself can be measured as:

$$p = \frac{1}{N_r}\sum_{i=1}^{N_r} b_i \tag{2.2}$$

where N_r is the total number of CRPs and b_i is the state of the ith response in the CRP dataset. Noting that the ideal value of p is 0.5 for when we observe exactly half of the CRP dataset have response = 0 and the other half have response = 1. Consequently, the ideal value for H will be 1.0. A highly randomized CRP characteristic in turn will have $H \approx 1.0$. This in turn means that the PUF characteristic is not biased towards a certain Boolean state. On the other side if the randomness is not high enough, it will mean that the output of the PUF is more guessable which in turn makes the PUF vulnerable against certain types of attacks. We will speak more of it later.

Uniqueness

Uniqueness is a measurement of PUF CRP characteristic which represents how unique the PUF instance is with respect to other PUF instances [11]. Noting that this measurement is done only on a group of PUF which have the exact same implementation. This means that PUFs' structure follows the same implementation blueprint on the exact same spot over each silicon die. The measure of uniqueness then assures that no two PUF instance have a similar CRP characteristic. This measurement is represented as in the following. First we feed all PUF instances with a unified challenge set. Then we measure the uniqueness of the response for every given challenge individually. To do that we measure:

$$U_k = \frac{2}{N(N-1)}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\frac{HD(ID_i, ID_j)}{L} \tag{2.3}$$

where N is the number of PUF instances, ID_i is the PUF instance i and ID_j is the PUF instance j, and L is the size of the response bit-vector. We do this for all the challenges. Then we will measure the average uniqueness as in the following:

$$MU = \frac{1}{K}\sum_{i=1}^{K} U_k \tag{2.4}$$

where K is the total number of challenges. Here we also expect the value of the uniqueness to be a normalized value between 0 and 1.0. The ideal value for uniqueness is also 1.0 like randomness. However, unlike randomness, which is measured for each PUF individual, the uniqueness is a measurement which is regarded with a group of PUF instances. This means that we need more than one PUF to evaluate the uniqueness.

Diffuseness

Diffuseness is the measure of how different the set of responses from the same PUF instance to different challenges are [10]. In other words, diffuseness measures the scattering of the 0s and 1s in a format that no pattern is recognizable from the conjunction of response values at any location in the CRP dataset. We can measure the diffuseness as in the following:

$$D = \frac{4}{K^2 \times L}\sum_{l=1}^{L}\sum_{i=1}^{K-1}\sum_{j=i+1}^{K}(b_{i,l} \oplus b_{j,l}) \tag{2.5}$$

where L is the size of the response bit-vector, and K the number of CRPs in the dataset. Diffuseness is normalized between 0 to 1 and the ideal value for it is 1.0.

Reliability

We already explained that PUF is not a source to represent a digital value with 100% reliability, although that is the expectation for an ideal PUF, practically the reliability of PUF is a value close 100% but not equal. We can measure the stability of the response given for each challenge in multiple acquisitions and refer to it as the reliability of the CRP dataset [10]. To do so, we observe the number of times a response is flipped for multiple acquisitions for the same challenge. The following is how we measure the Reliability:

$$S = 1 + \frac{1}{N_c}\sum_{k=1}^{N_c} log_2\, max(\frac{\sum_{j=1}^{N_a} b_{k,j}}{N_a}, 1 - \frac{\sum_{j=1}^{N_a} b_{k,j}}{N_a}) \tag{2.6}$$

where N_c is the number of CRPs in the dataset, and N_a is the number of times the challenge is used. The reliability is set between 0 to 1.0 and the ideal value for it is 1.

For all the metrics presented above, a good PUF which can represent values close to the ideal value for each metric is considered a good PUF. With that given then we explore other aspects of a PUF, such as the complexity of the structure of the PUF which is closely related to the feasibility and security of the PUF.

2.4 PUF Structure Variations

As mentioned earlier in this chapter, PUF has two major categories, the Weak PUF and the strong PUF. Each category also has many families and sub-variants. This is predominantly true about the strong PUF family. Here we will elaborate on some of the PUF families in the category of strong PUF and the several from the weak PUF family.

2.4.1 Arbiter PUF and its sub-variants

Arbiter PUF is one of the common strong PUF structures that exist to this day. A simple yet largely capacitated structure that can produce very large amounts of CRPs. Arbiter PUF is conceptually based on creating a racing condition between two signal paths. These two signal paths are symmetrical and run through multiple stages that are sequentially connected to each other. Each stage has two options for both signal paths to either run in parallel or run across each other. The decision on that is made by a selector bit which is in control of the user of the PUF. The end point for the racing signals is an Arbiter which receives one signal on its SET input and the other on its RESET [12]. The concept of arbiter PUF assumes that at each stage, due to the micro process variations, the signal paths will pick differences in terms of their speed. In other words, due to the variation, it is possible that at one stage one signal goes through a shorter path that the other, or vise versa. This variation is then accumulated into the propagation delay of each signal after passing through each stage. At the end, it is then expected that one signal reaches the arbiter faster than the other, and then defines the output of the arbiter. A block view over the structure of a simple arbiter PUF is shown in Figure2.2. Noting that such micro variation in propagation delay is not just internal, but also should exist when two PUFs with the same structure and from the same device family are compared to each other. This technically means that the signal propagation delay differs from device to device, considering a subset of challenges, which then results to different response for each challenge when the responses are compared from

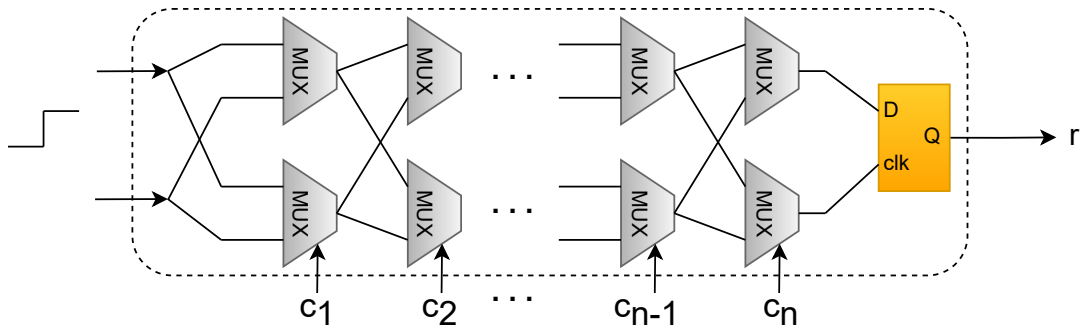


Figure 2.2: Illustration of the structure of an Arbiter PUF.

device to device.

We can see here that if we increase the number of stages, we can expect to have exponentially increased the number of possible racing path combinations. This in turn means very large number of CRPs. However, the structure of Arbiter PUF itself is not secure on its own and can be modelled/digitally cloned very easily. To avoid that, several variations of it have been introduced over the past decade and half. Some of these variations are:

- XOR Arbiter PUF: When the output of all Arbiter chains is XORed to represent the final output [13].

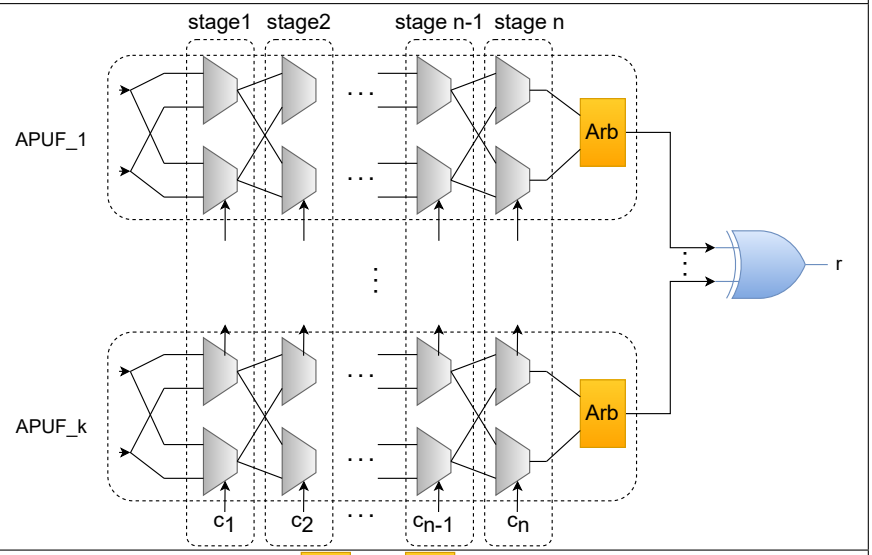
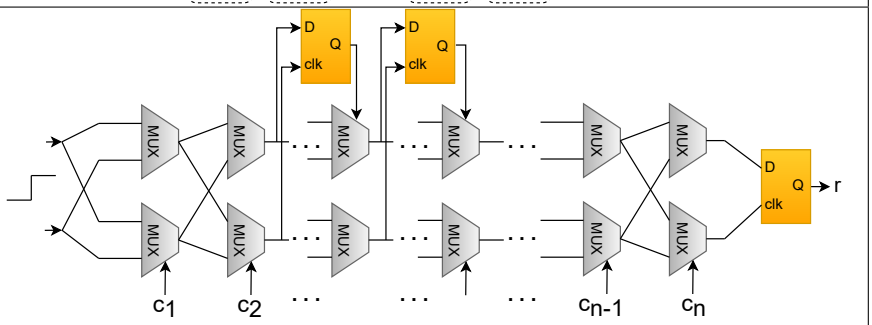
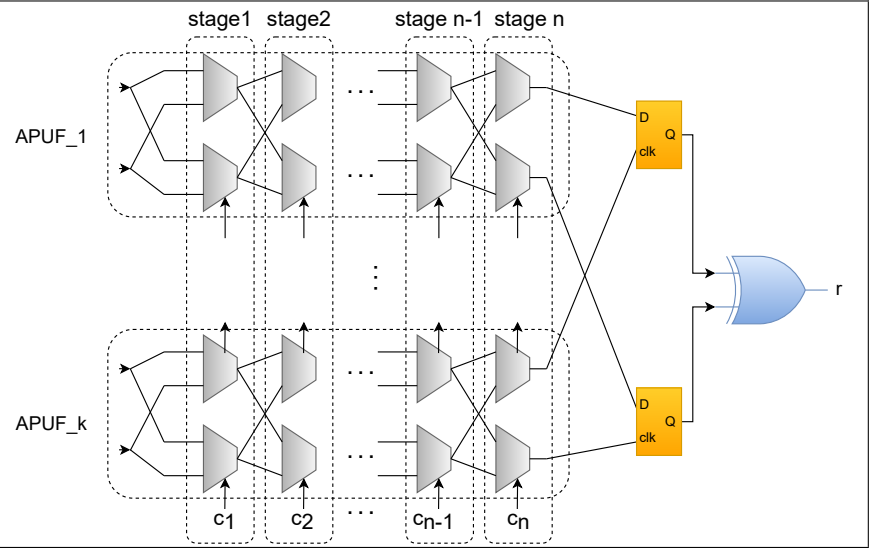
- Feedforward Arbiter PUF: When the output of one or some arbitrary stages are connected to the input of one or some other stages inside the same arbiter chain [13], [14].

- Double Arbiter PUF: When there exist multiple Arbiter chains and the idea is to setup arbiters to the number of existing possibilities of each two racing paths from two and bottom of each arbiter chain, considering all the arbiter chains [15].

The Table 2.1 shows the structure of the mentioned arbiter PUF variants. These variants nowadays can promise a more secure arbiter PUF implementation, which in turn gained more attention on the Arbiter PUF family for industrial implementation.

Arbiter PUF and its variants are of the main research focus in this work, that is why we elaborate more on this family of PUF. However, there are other family of PUFs in the weak and strong categories, which we will elaborate on briefly in the following.

Table 2.1: Structure of 3 variants of Arbiter PUF

Arbiter PUF variant	Structure
3-XORed Arbiter	
Feedforward Arbiter	
Double Arbiter	

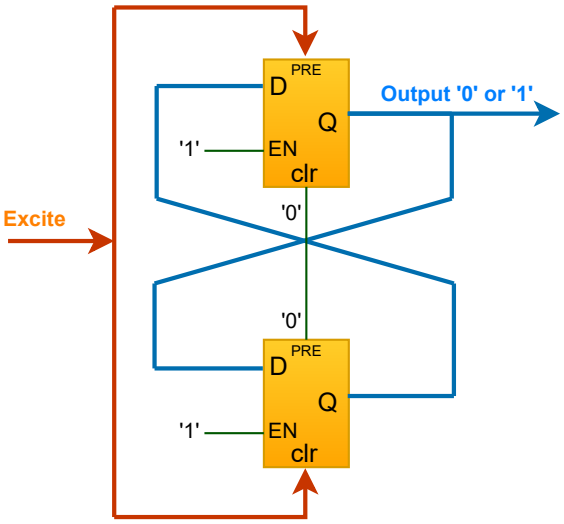


Figure 2.3: Illustration showing the structure of a butterfly PUF.

Butterfly PUF

The concept of the Butterfly PUF is based on the idea of creating structures within the FPGA which behave similarly to an SRAM cell during the startup phase [16]. A Butterfly PUF cell is a cross-coupled bistable circuit, which can be brought to an unstable state before it settles to one of the two stable states that are possible. The structure consists of two latches whose outputs are cross coupled as indicated in Figure 2.3.

Ring Oscillator PUF

This Ring Oscillator PUF is composed of many delay loops that oscillate with a particular frequency. They are laid out identically, but the minor variations in manufacturing lead to loops with slightly different frequencies. The loops drive counters which are used to produce the response bits to a given challenge. Figure 2 shows the structure of this type of PUF [17].

Memory PUF

From the notion of PUF, we inherently believe that it is a designated functional unit that is distinguished from the rest of the SoC components for instance in a silicon chip. PUF however can be seen as a special behavior observed at a certain state in a conventional SoC component. An example of that is the memory PUF. We know that any SoS

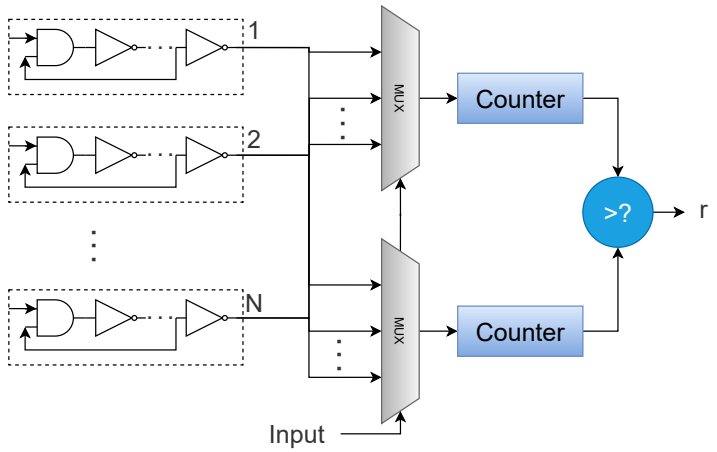


Figure 2.4: Illustration showing the structure of a ring oscillator PUF.

nowadays has at least a small on-board memory for run-time data exchange. An SRAM component can be seen as a unit used for that case. Now, it is observed that when an SRAM is powered up, at the early stage of the power-up, before a whole memory wipe-out procedure is called, every memory cell randomly has 1 or 0 state. This state also, for every memory cell, stays the same (ideally) for every power-up. Yet there is no expectation on what memory cell has what binary state when the power-up values are seen for the first time. This means that the 0 and 1 values are randomly scattered in-between the memory cells. Such behavior in an SRAM is also called a PUF [18]–[20]. Now, as mentioned, it is expected that what we call PUF has an input, hence a function. For an SRAM, we can consider the memory address being the input, or challenge, and the series of random 1s and 0s we get for that specific memory address, can be considered the response. Such randomized behavior of SRAM also is thanks to the inherent differences between the voltage thresholds in each memory cell, which is normally designated to each cell during the manufacturing process as a manufacturing error which is inevitable. Such error of course is correctable with an initial zeroing process when the device is turned on. But before that, we can use these values and consider the SRAM itself at that stage, a PUF.

Bistable Ring PUF

The idea of Bistable Ring PUF was first introduced in [21]. The basic idea of the BR-PUF is based on the fact that an inverter ring consisting of an even number of inverters has two possible stable states. Similar to a static random-access memory (SRAM) cell which

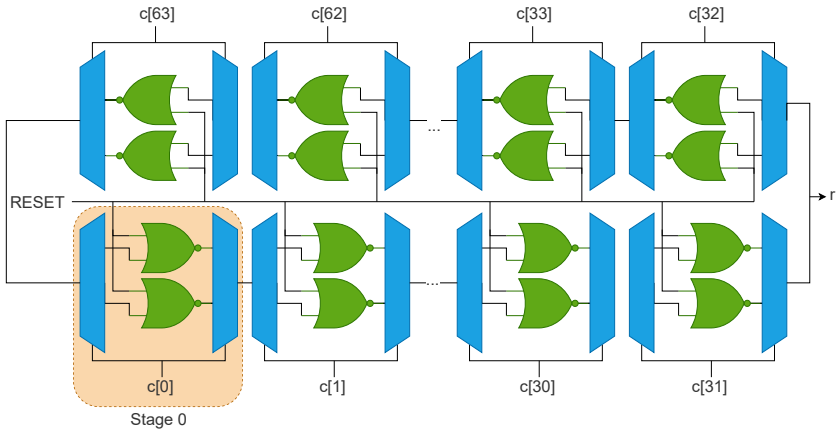


Figure 2.5: Illustration showing the structure of a ring oscillator PUF.

is based on a pair of cross-coupled inverters, a ring with even number of inverters falls into one of its two possible stable states when powered up or, more generally, when it is released from some unstable state. We call such kind of an inverter ring a bistable ring. Figure 2.5 shows the structure of a 64-bit bistable ring PUF.

Optical PUF

Optical PUF is perhaps one of the primary representations of PUF. An optical PUF which was termed POWF (physical one-way function) [22] consists of a transparent material that is doped with light scattering particles. When a laser beam shines on the material, a random and unique speckle pattern will arise. The placement of the light scattering particles is an uncontrolled process and the interaction between the laser and the particles is very complex. Therefore, it is very hard to duplicate the optical PUF such that the same speckle pattern will arise, hence the postulation that it is "unclonable".

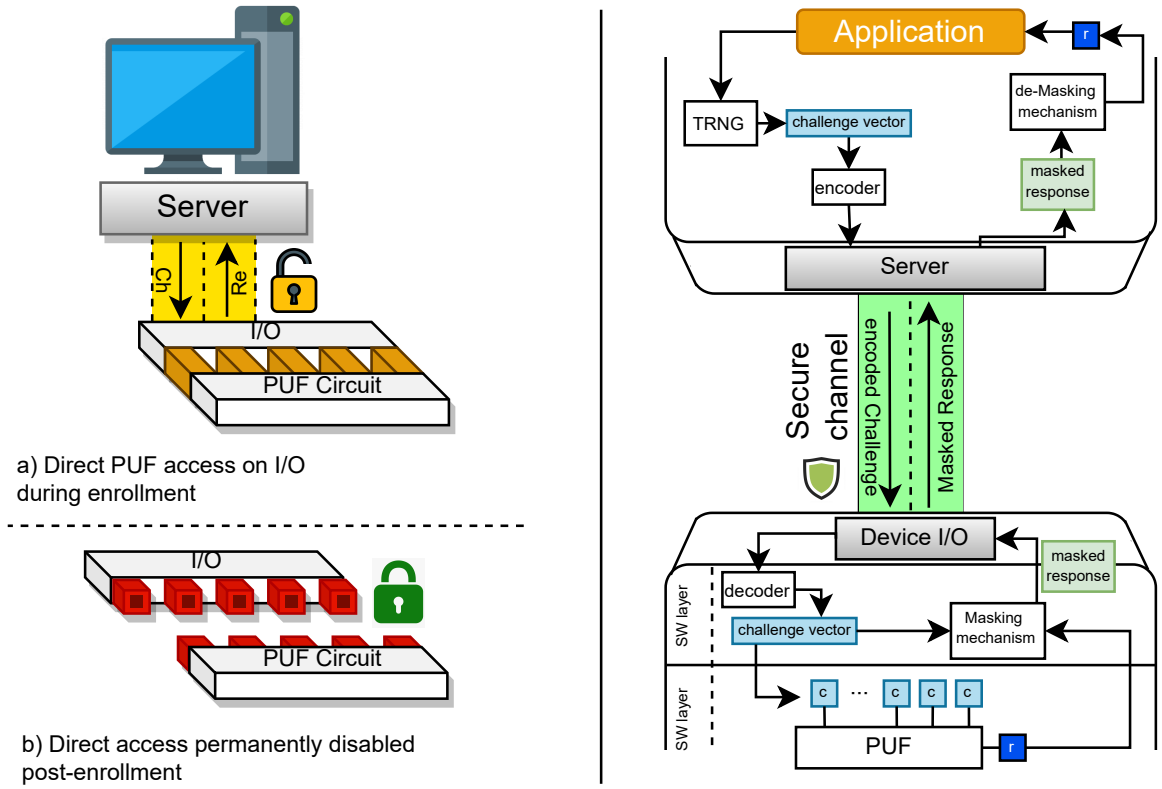
2.5 Conventional PUF Protocols

Despite the definition of application for PUFs, we can independently see PUF as a security primitive and assign it to encryption and security protocols. We can then designate the protocol to the appropriate applications knowing the potential that the protocol has for the security aspects of the application. Here we point out to some of the well-known security protocols that use PUF as their security primitive.

2.5.1 Fundamental steps of using PUF

A primary step in preparing PUF for cryptographic or any security-based protocol utilization, is to enroll the PUF. The process of enrolling the PUF comprises steps to take in order to collect and post process CRPs from all the manufactured PUF instances. An operator is in charge of preparing the just-manufactured devices for deployment, by connecting the PUF-enabled device to a collector machine who is either itself a server or is connected to a server that is storing the PUF CRPs. We expect that during enrollment there are direct I/O connectors to the PUF to facilitate the CRP read-out (See Figure 2.6 (a)). This in turn gives the operator to rapidly collect the CRPs without passing through unnecessary connectors. While at this state, we assume also that the connection between the collector machine and the PUF is safe and secure. Once the CRP collection is complete, then the direct connectors to PUF and the system I/O are eliminated (See Figure 2.6 (b)). Normally for this the designers allocate E-Fuses that provide such one-time connection. The disabling of the direct I/O to PUF is expected after the CRP readout to protect the PUF from unwanted queries from the outside of the device. So, the first level physical security is provided as such.

Then the PUF is available for mission mode, where it is used for any cryptographic or authentication task. In such setting, The PUF is directly invoked only by the device that is housing the PUF. Thus, the demand for PUF CRP is given first to the PUF-enabled device, and then redirected to the PUF after the request is passed through some encoding and decoding mechanisms. The response of the PUF is then received and ready to be sent to the demander. It is expected that the PUF response is sent securely to the demander. This can be assured in two ways. Either the designer makes the PUF complex enough so that the relation between the challenges and the responses of the PUF is not exposed, therefore the exchange CRP shows a totally random behavior, or we expect that the response is first wrapped into some post-processing procedures and is presented in an obfuscated or masked form to the demander from outside of the PUF-enabled device (See Figure 2.6 (c)). This level of security then corroborates the safety and security of the communication channel. Later on, in the manuscript we elaborate on why such security is required. But generally explaining, the obfuscation of requests for PUF response (a package comprising the challenges to the PUF) and the PUF responses is needed to prevent attackers, especially those that use machine learning techniques to create a digital clone of the PUF. Having access to the raw CRP of the PUF can give attackers such capability which in turn compromises the PUF in a sense that the at-



(a) Physical access (b) Remote communication channel

Figure 2.6: Illustration showing the restrictions in the physical access in enrollment mode, and the secured communication channel during mission mode.

tacker then can impersonate the PUF to request for confidential data that is only for the PUF-enabled device to read. In the following we elaborate on two common forms of PUF utilization, the authentication, and the key exchange.

Authentication

PUF-based authentication protocols are used commonly to identify that the person or the device that is communicating with a verifier server is authentic and original [23]. To do so, the sever first collects an adequate number of CRPs that represent the device's fingerprint(s) and stores them on a database correspondingly to the User's or device's identification (See Figure 2.7 (a)). Later on, When the device is needed for authentication, the Server itself, or a trusted third party (TTP) identifier will ask the device's owner or itself for a fingerprint (part of or the same that has been collected during the

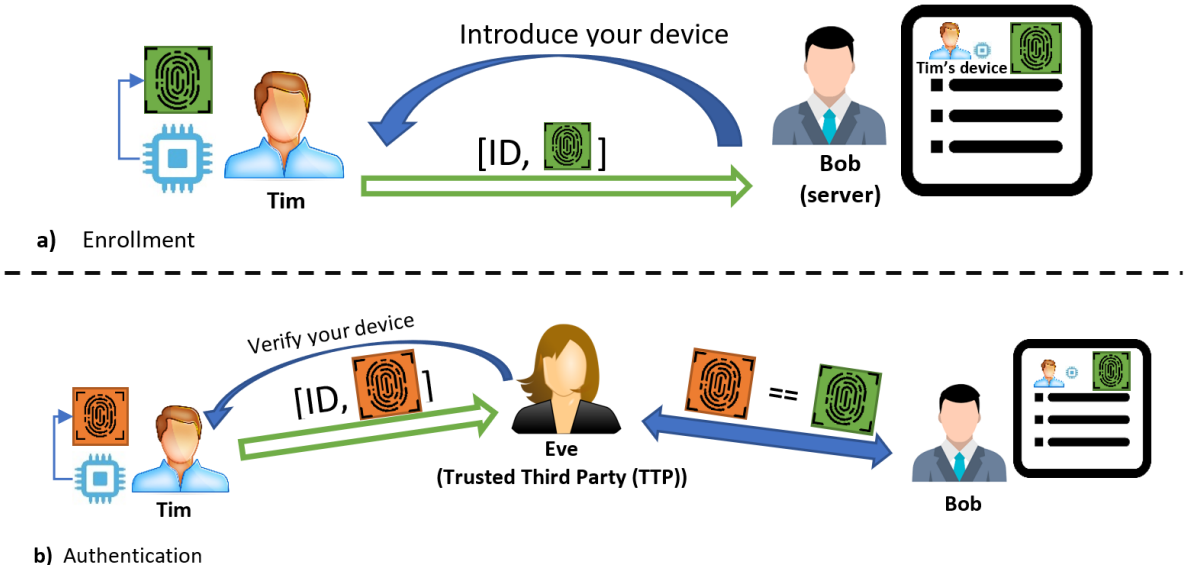


Figure 2.7: Illustration showing the enrollment and mission modes for an authentication protocol.

enrollment). Once the fingerprint is fetched, it is compared to its similar counterpart that has been stored earlier in the enrollment phase. If they are equal (fractionally or exactly), then it is decided that the device is authentic (See Figure 2.7 (b)).

Key Exchange

Contrary to authentication protocols, in key exchange protocol the idea is to encrypt the communicated message between two or more parties. In such case, commonly there is a centralized unit which synchronizes the key exchange between the parties, including itself. Similarly, for exchange an enrollment phase is needed. In this phase the CRPs of the PUF are collected and sent to the server for storage as encryption keys. Noting that the server creates code-offsets of these keys as well and store them [24]. Code offset is a value extracted from the PUF response during enrollment, and is later used to recover the same response on the device's side (See Figure 2.8 (a)). The code offset is useful in this case to recover original responses from noisy responses during mission mode. As we discussed, some PUF responses have a chance to flip their binary value. This is called the instability of the response. While in a good PUF it is in not considerable to cause certain issues, yet for encryption key generation, it is a necessity to mitigate it to ensure that we create the same key as the one that is used to encrypt the message. Figure 2.8 (b) shows the process of key exchange and a simplified representation of key

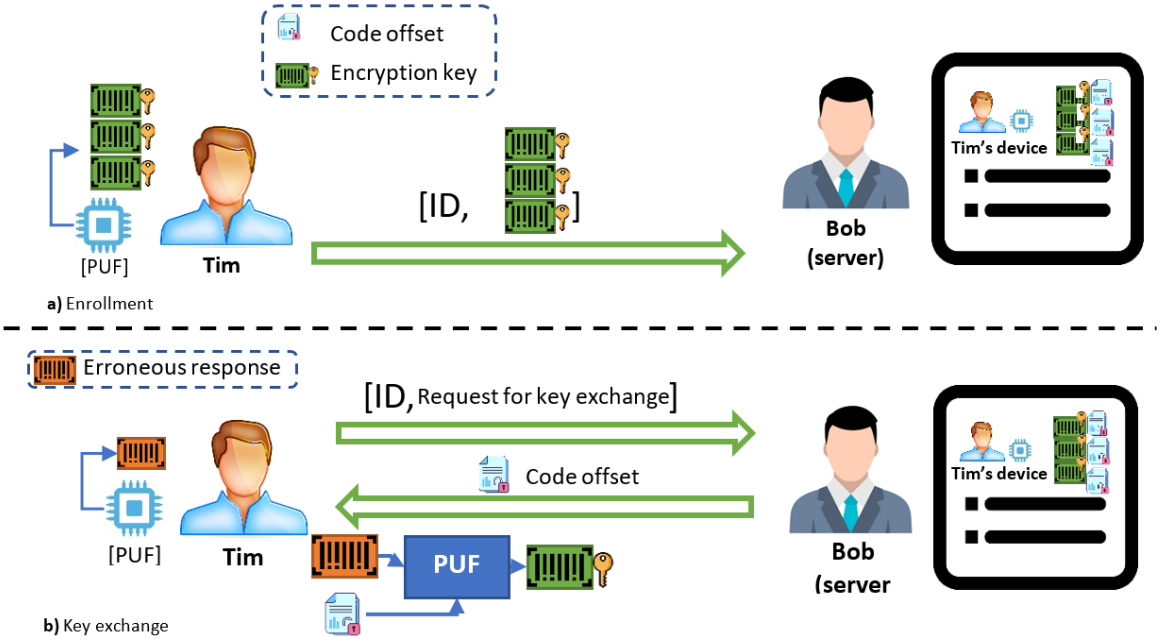


Figure 2.8: Illustration showing the enrollment and mission modes for a key exchange protocol.

recovery on the PUF-enabled device's side.

Up to now we know how two of the most common use-cases of PUF work. We explained the authentication and the key exchange, their general schematic and what they require to operate. Later on, we show that there exist some companies that have actual products with PUF enabled inside for cryptographic purposes. After that we begin elaborating on some of the notable challenges that we need to deal with while considering PUF. This then opens up the road to where we take inspirations and motivations for the core contribution of our work which we elaborate in the last subsection of this chapter.

Existing Commercial PUFs

PUF of course is not a concept belonging only to research-oriented works. While many aspects of it is being studied to this date, the concept of PUF itself goes back to two decades ago. Since then, of course, several companies and research institutions have established solidified security conceptions that comprise PUF as one of their constituents. In Table 2.2 we point out to several known commercialized PUF products.

Table 2.2: Semi-conductor solution companies and their PUF-based protucts

Index	Company Name	Products
1	Intrinsic ID	Butterfly PUF: Apollo FPGA IP (hardware IP) SRAM PUF: QuiddiKey IP (hardware IP)
2	Enthentica	FPGA PUF IP ASIC PUF IP
3	PUFSecurity	PUF as root of trust (PUFrt) PUF as part of Crpyto Processor (PUFcc) PUF as a Secure Element (PUFse)
4	NXP	Multiple PUF-based solutions MCUXpresso SDK API
5	Secure-IC	Securuzr PUF as a root of trust element

2.6 Challenges, Opportunities, and the Motivations

Despite the promising potential the PUF has and the variety of applications it is favorable to, there are several challenges coming with it as well that need resolving. Below are some of the common ones:

1. Instability of PUF response in normal operational mode
2. Sensitivity against aging and environmental variations
3. Vulnerability of error correction codes against manipulation attacks
4. Vulnerability against machine learning based modeling attacks
5. Incapability of conventional storage methods of the very large CRP dataset

1: One of the main challenges with PUF is the instability of the response. What happens in that nature is that when PUF is queried with a challenge, there is chance that the corresponding response is altered through multiple queries. Although it is essential to ensure that this probability factor is low. Nonetheless, for some challenges there exists an inevitable high chance of the response to show an alternate value.

2: Sensitivity against aging and environmental variations is another vulnerability the PUF devices have. Normally the response alteration is amenable, at least for the PUF that are accepted as workable functions before being deployed (during the enrollment phase for instance). But even in that nature, when the PUF-enabled device is exposed to some extreme environmental settings, it will show reflections in way that its CRP

characteristic is considerably shifted. To avoid such situations getting in the way, PUF-enabled devices are tested in different environmental situations during enrollment and their CRP is collected according to each setting so that server will show some adaptability to such situations. This is true also for when the PUF is utilized for a long time and the aging factor affects the CRP characteristic in a way that again a considerable portion of it is shifted to an altered manifestation.

3: Instability of PUF though is amenable with error correcting code (ECC) and fuzzy extractors, which are special coding/decoding (codec) that facilitate PUF original response recovery through code offset sharing (also known as helper data). As we explain later in future chapters, this type of error correction mechanism is vulnerable to manipulation attacks. How it works is that attackers gain access easily to the code offset, and by manipulating the value of it and redirecting it to the PUF-enabled device, they drive the recovered response into a value that is easier to guess and so they can recreate the key by some trial-and-error.

4: On top of the issues the PUF instability causes, and the amount of exposure the publicly available error correcting helper data values provide for potential attackers, there exists also the chance that PUF can be modelled by exploiting the power of machine learning. This happens when the Challenges and responses are publicly exposed on communication channels. Although, a key PUF design factor is to make the PUF complex enough so that modeling it is either not possible or requires considerably large number of CRPs usually in the order of millions to make it possible to model the PUF. But as the PUF design is getting better through time, the machine learning modeling techniques also get better over time and so both lanes of evolution run in a competition that is one picking after the other.

5: One of the key potentials of PUF is that it avails a very large CRP space to create encryption keys and fingerprints from. This is true mainly for the strong PUF designs that have a large input challenge size. The acceptable challenge size is usually above 64-bits. This in turn gives us 264 responses which is an astronomical number and practically impossible to save into a storage device. Even if possible, given that such number of CRPs is for just one PUF instance, and we can expect the same for all the instances, therefore it is not possible to save all the CRPs if the approach is to just suffice to build a CRP dataset.

Given the above-mentioned challenges, it is possible to deal with them potentially if we consider using Machine Learning and PUF itself as a duo to create authentication

fingerprints and encryption keys. Such potential builds the core motivation of this work. In the following we explain how this potential exists and how we can cultivate it.

2.7 Machine Learning and PUF Modeling

Machine learning is an iterative process to create an estimated solution for a problem, based on a set of data. It is assumed that the dataset that the machine learning creates a solution according to, has a patterns or features that consistently appear in the data samples. The Machine learning solution in turn is set to associate sets of features to data with low dimensionality. An example of that is when a model is created to classify handwritten digits from imagery data. In such case, the features are the lines in different forms, such as straight line in different angles, or curve lines, and the association of variations of these lines appoint a specific digit. Machine learning solutions are not just for classification. ML methods can create regression solutions as well, where the output is a continuous value. A simple example of that is when an ML solution is created to estimate the output of a math function. ML in itself includes a training loop. An iterative process that adjusts the internal status of the model to the relationship seen between the input and the output. In the training process, for one batch of data samples, the model will do an inference first, and the outputs of the model are assessed for that batch based on its distance of the model's estimation to the correct output. Depending on how far the inferred output of the model is, a loss value is calculated and a fraction of it is propagated back to the model to adjust the internal values. Noting that the internal values of the model can be considered as weighted connections between different internal states, or in case of linear problems, just the connection between the input and output. The idea of adjusting the values during training is to modify their value so that based on the input, the cascading flow of state change will lead to the correct output. The training loop runs for one epoch meaning that all the data samples of the training dataset are met once, and the loop will run multiple epochs until the loss value is minimum and correspondingly the accuracy of the estimation of the model is converged to a maximum probability. Given that the estimation accuracy of the model also can be assessed in different ways. For instance, for classifier models, the proportion of correctly classified data samples to the total number of data samples classified in a dataset that the model did not see during the training.

Modeling PUF using machine learning is the most facilitated way of digitally cloning

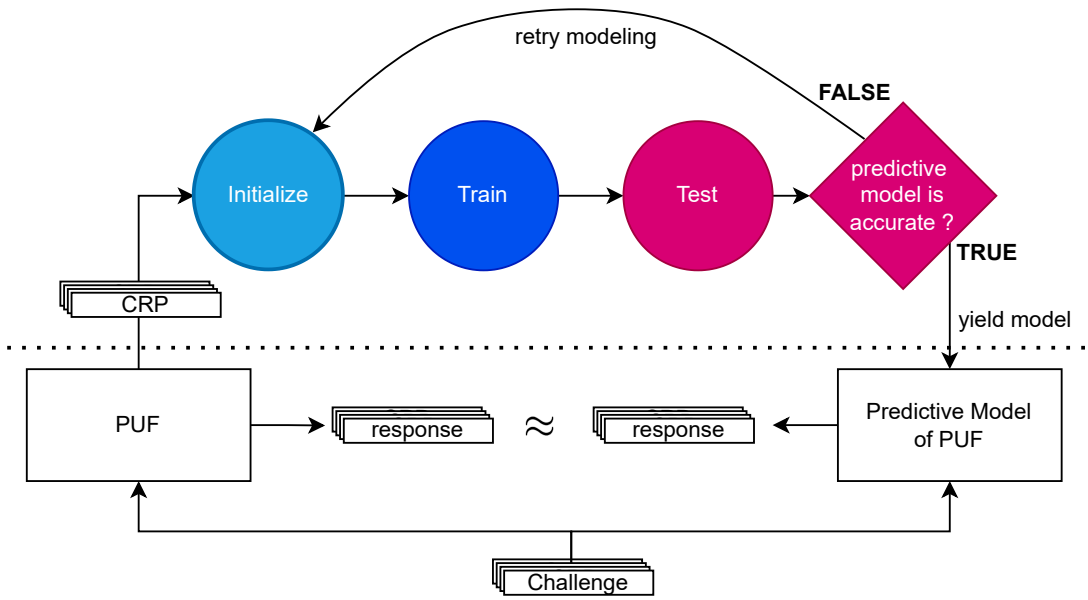


Figure 2.9: Illustration showing the procedure of training a predictive model of PUF using machine learning

the PUF. The way it works is that we collect a dataset of CRPs of the PUF, and we send the dataset to a training process, and run the training for multiple epochs until we obtain a model solution that can accurately predict the response of the PUF for any given challenge. This procedure is also depicted in Figure 2.9.

Multiple parameters play important rules in increasing the probability of obtaining an accurate model. For instance, in the process we can define how we optimize the model. So far, there are multiple optimization methods introduced, such as stochastic gradient descent, adaptive gradient descent (AdaGrad), Adam optimizer, and RMSProp optimizer. Each of the optimizer functions apply differently the loss value back to the model. We also have different methods in calculating the loss value. For instance, we have cross entropy loss, and binary cross entropy loss. Cross entropy loss measures the difference between the discovered probability distribution of a machine learning classification model and the predicted distribution. All possible values for the prediction are stored so, for example, if you were looking for the odds in a coin toss it would store that information at 0.5 and 0.5 (heads and tails). Binary cross entropy loss, on the other hand, store only one value. That means it would store only 0.5, with the other 0.5 assumed in a different problem, if the first probability was 0.7 it would assume the other was 0.3). It also uses a logarithm (thus "log loss").

Aside to the optimizer and the loss function, there are other parameters exist that

can be modified and affect the training, such as the batch size which indicates the number of inferences analyzed in one take and their loss values are combined to then be propagated back to the model, learning-rate which indicates the proportion of the loss value that is to be added to the internal values of the model, and the model topology. However, the model topology has different parameters, such as what is the activation function of each internal state of the model, and the connectivity of the internal states to each other. In the following subsection we elaborate more on these parameters.

2.8 What Predictive Model to Use

Predictive models made to cooperate with PUF can be presented in different ways, depended on what is the application or problem to solve. Some of the known ones are PUF circuit classifier and PUF challenge classifier. The PUF challenge classifier is the most known approach in modeling PUF. The idea in this approach is simply to create a predictive model that predicts the response for any given challenge. Using an appropriate model structure is a critical decision as we know. Knowing the model structure and hyper-parameter values beforehand which can achieve the desired prediction accuracy with plausible number of training samples is a big advantage. We had that since we started this research by looking at some of the already published works. To the best of our knowledge, machine Learning in itself presents a broad field of predictive and estimated solutions. One of the key considerations so is to do the research and find out what model structure can be used to obtain the desired outcome or beyond. The outcome can be mainly expressed in terms of the prediction accuracy. However, often times the outcome comprises the price paid to prepare the promising training conditions, such as the amount of training data, the time of training and number of trial and errors, and the computation power. A relevant point that can help finding the proper model structure is the complexity of the training subject. Here we are focused on modeling PUF, therefore the complexity of the PUF structure can imply what ML model structure we should choose. Below we elaborate on variations of ML models that have been used so far to model PUF.

2.8.1 Logistic Regression

Logistic Regression is an ML model solution that is suitable for classification problems with linear relation between the input and output [25]. It's counterpart, Linear Regression, exists also which is a different model solution and is majorly targeted for regression problems. A simple definition of Logistic Regression (LR) as a solution is a sum of weighted inputs that are passed to an activation function to yield a probabilistic value between 0 and 1. It can be written as:

$$Z = \sum w_i x_i + \beta \tag{2.7}$$

where w_i is the ith coefficient value for x_i the ith input parameter, and β is the bias value. The activation function can also be written as:

$$S(z) = \frac{1}{1 + e^{-z}} \tag{2.8}$$

As mentioned, the output $S(z)$ is value between 0 and 1 (a probability estimation), and e is the base of natural log (also known as Euler's number).

There are several works that have proposed modeling PUF with LR. And the LR is used predict the relationship between the challenge and response. For instance, the work of Ruhrmair in [26] proposed modeling Arbiter PUFs with LR. Their evaluation also showed that using LR to model APUF can yield a model with above 99% prediction accuracy using a small set of CRPs, usually in the order of several hundreds.

2.8.2 Support Vector Machine

As discussed above, LR is suitable to make predictive model solutions for problems where the relationship between the input and output is linear. Arbiter PUF as we said was an example of that. However, when we increase the complexity of the problem, hence adding non-linearity to the input/output relationship, LR cannot provide an accurate solution. An example of that in modeling PUF would be to create a predictive CRP model of an XOR Arbiter PUF with any number of XORs above 2 [26]. In this case, solutions designers have found out that a more sophisticated modeling strategy called Support Vector machine (SVM) can potentially create a predictive model solution. SVM is a modeling strategy similar to LR, with the difference that SVM attempts to create a plane that separates the classes, rather than a line [27]. To do so, SVM project

the input space into a new space with more dimensions and adjusts a plane separator (hyperplane) to classify the domains according to the placement of samples in the new space.

In the case of modeling PUF, several works proposed modeling XOR Arbiter PUF and Feedforward PUF using SVM [26], [28]–[31]. SVM can be considered a viable and rapidly trainable solution to model PUF with moderate structure complexity. However, studies have shown that more complex PUF designs are resilient against modeling using SVM. Therefore, SVM can be reserved a solution to just a subset of strong PUF variants. Moreover, SVM is not resilient to noise. We will speak about this later on that PUF instability can affect the training results, and in terms of SVM, it is an effective factor that can potentially downgrade the prediction accuracy.

2.8.3 Decision Tree and Random Forest

Decision tree (DT) is also an ML solution for classification and Random Forest (RF) is a modeling strategy based on DT in ML to create model solutions more accurately for non-linear problems. We will first start with DT. Like LR models, DT model solutions can represent regression solutions or classifier solutions [32]. Thus, if the end goal is to predict a discrete value for the output of the model, then the solution will be a classifier. However, DT solutions can solve non-linear problems unlike LR solutions. The main advantage of a decision tree is that it adapts quickly to the dataset. This gives leverage to DT compared to SVM, as it can solve no-linear problems better without needing extra pre-processing. Moreover, DTs are better at solving categorical problems and can deal with the problem better colinearly compared to SVM.

Topologically, a DT classification model solution has leaves which represent the predictable values for the output, decision nodes which outlie to two or more branches, and branches which represent the conjunction of features that will lead to the prediction of the output. Nonetheless, there are debates on the accuracy of DT classifiers. For instance, DTs can easily incorporate outliers to the underlying model within the dataset. Of course, we can consider noises creating such outliers which are not of the interest. Therefore, DTs will face challenge in modeling real PUF since there as we said exists unstable CRPs even within PUFs with good characteristic.

In such case, Random Forest (RF) can be a good candidate to provide a model solution. RF models in fact comprise multiple DTs whose output for classification is sent for an averaging function or a voting function to yield the final classification output [33].

RFs in turn have more accuracy than DT due to the fact that they incorporate several DTs each trained with a portion of the training dataset or with a different random initialization state. This gives the RF the ability to deal with the noisy data better than DT. Nonetheless, such mechanism in RF builds complexity in the solution which makes it more resource hungry and difficult to interpret and visualize [34]. Nonetheless, due to the success that RFs brought in modeling PUF, it has been discussed in several works in the literature [35]–[37]. Therefore, RF can remain as a viable modeling solution for PUF modeling.

2.8.4 Multi-layer Perceptron

Multi-Layer Perceptron (MLP) is a variant of Artificial Neural Networks (ANN) that have been used for a broad range of applications and used for modeling PUF in specific and showed potential results. ANN have many variants of itself, each of them distinctive due to their unique structural organization. To know to what MLP is, we need to know what ANN is. ANN is a model solution which in itself comprises an interconnection of nodes in different layers. Topologically, we can see ANN as a network of nodes, wherein nodes are ordered in different consecutive layers, and nodes of a layer are connected to the ones in other layers. An illustration of MLP which is an example of an ANN is shown in Figure 16. Now, how the topology of an ANN is designed, will speak of its variation. For the MLP, the topological feature is that nodes in one layer are connected to all nodes to the next layer (fully connected), and all the connections are set forward, thus no node in a proceeding layer is connected to a node in a former layer. The input nodes are nodes that accept the input parameters, and the output node(s) represent the classes that the model is anticipated to predict.

The organization of an MLP is considerably different than that of a RF, or SVM. The nodes in an MLP (or ANN in general) are called neurons, and each can be considered a feature extractor of a data that is being given to it [38]. The connections between neurons also are called synaptic connections or weighted inputs to the neurons. As expected, the weights are values that will be multiplied on the data passing through that connection. We consider each neuron as a linear classifier (a perceptron), thus it will have a bias value and an activation function (See Figure 2.10). The interconnection of these linear classifiers so will form a non-linear classifier. Therefore, an MLP can in turn be expected to solve prediction problems with non-linear relationship between the input and the output. MLP is an expandable model also, meaning that there is no cap in

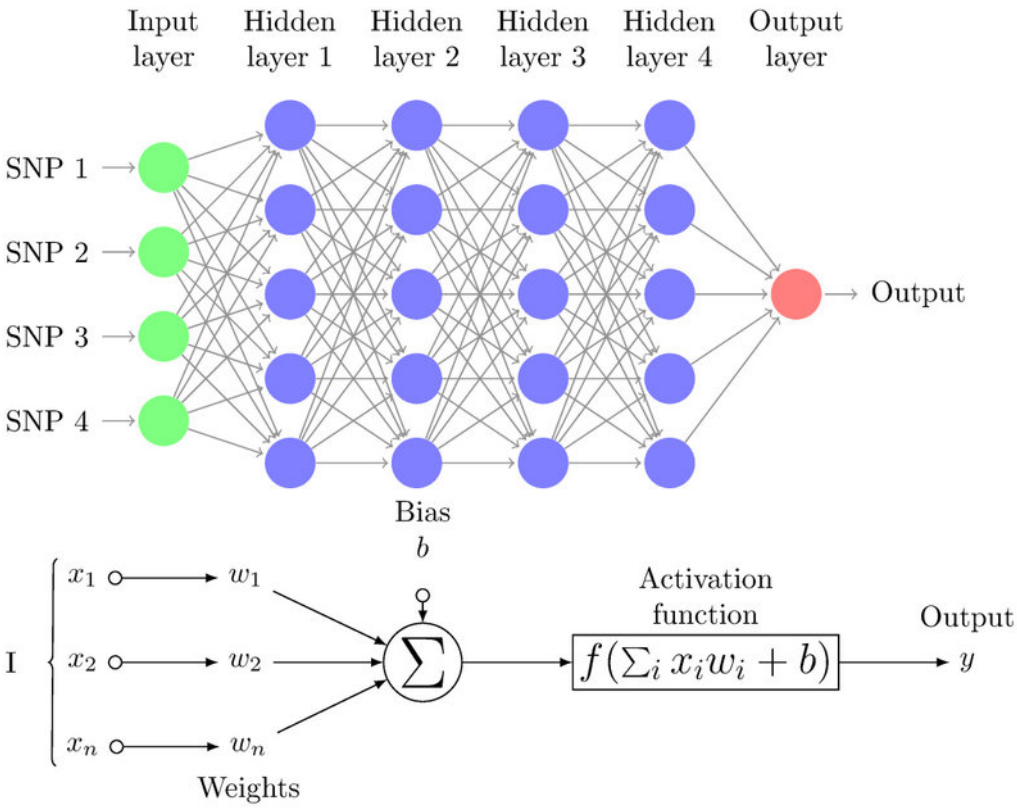


Figure 2.10: Illustration showing the structural organization of an MLP, and the computational format within a single neuron which represents a single linear classifier (below).

the number of layers or neurons in each layer. Therefore, an MLP can grow in number of neurons or layers to fit into a solution for complex problem.

In modeling PUF, MLP has been used in several works such as in [39] by Ruhrmair and [40] by Alkatheiri, where they proposed using MLP to model XOR Arbiter PUF with large number of XORs. MLP seems so far as a potential model solution to create models of PUFs with increased structural complexity. However, MLP is a general solution model and for specifications, solution designers can modify the hyper-parameters such as number of neurons and layers to create models which can potentially require less training effort and data to achieve high prediction accuracy. Such work for PUF modeling has been done in the literature by Mursi et al in [6], where they propose adapting the structure of an MLP comprising 3 hidden layers, to the structure of the target XOR Arbiter PUF. In specific, their model follows the number of XORs of the

target PUF to decide how many neurons each of the three hidden layers should have. The number of layers however is fixed. Such model compared to its predecessors is less resource hungry and thus can be trained with a smaller number of training samples. Consequently, the model is trained faster than the other MLP models with large number of neurons.

2.8.5 Convolutional Neural Network

Despite their adaptability and potential in modeling PUF with increased structural complexity, MLPs are not entirely resilient to noise. Moreover, if the training data samples are dispositioned slightly, the training will highly likely converge to no solution with high prediction accuracy. This is true for other problem fields that suffer from the same deficiencies, including noisy data and dispositioned data. In such case, another variant of ANN model solution can be used which is called Convolutional Neural Network or CNN. CNN models are specific types of ANN that have a different organization in some or all of their layers. Aside to fully connected layers of neural networks, CNN has layers of convolutional neurons that act as feature extractors that convolute over the input data. In this organization, the input to each of the Convoluting neurons or feature extractors is a portion of the overall input data at a time, and the feature extractor moves over the entire input data until all parts of the data frame are met [41]. An example of a CNN model with 4 convolutional layers is brought in Figure 2.11. In the organization of a CNN, the fully connected layers come at the end part of the model, where they process the lateral features and provide data to the classifier layer at the end.

Compared to MLP, CNNs have more resilience to noise and dispositioning in the training samples. Moreover, CNN models can be more useful in class of problems where the classification at the end is depended on the spatial relation between the data points in the input parameters. Such problems are mostly found in image processing field. In PUF modeling however, CNNs are scarcely used. Since spatial relations seldom exist in the challenge vector which can affect the response at the end. Nonetheless, works exist that used CNN for PUF modeling, such as in [42].

2.8.6 Discusion

Given the variation of modeling solutions that has been discussed here, we can infer that in cases where the individual subject for modeling has a linear structure, such as

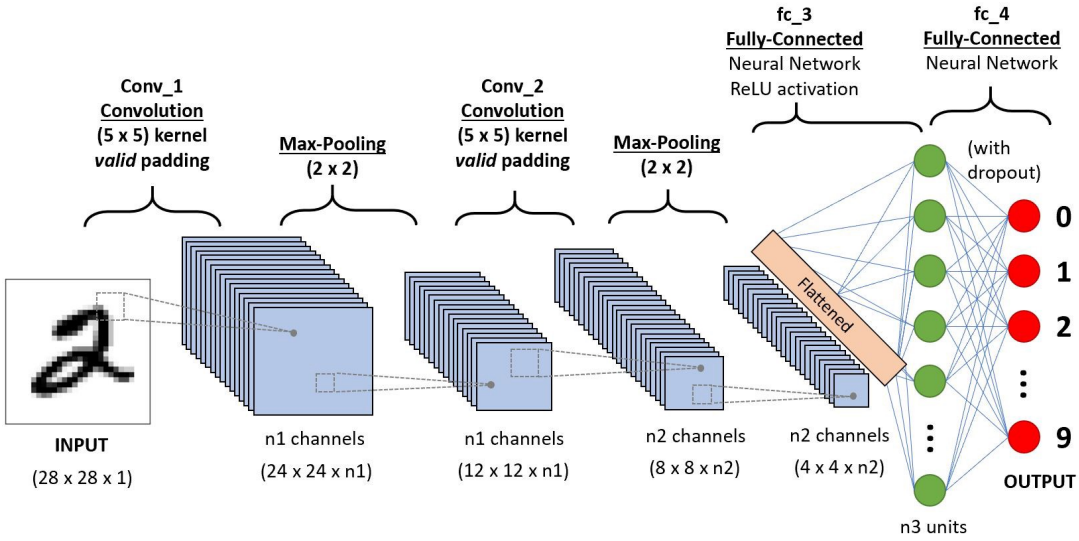


Figure 2.11: Illustration showing an example of a CNN model with 4 convolutional layers and 2 fully connected layers. This example model is depicted as a classifier of handwritten digits.

an Arbiter PUF, then logistic regression can be used. On the other hand, if the individual subjects to model are non-linear, solutions such as SVM, RF or MLP can be used. If the non-linearity is moderate, then SVM can be a good candidate to rapidly provide solutions with high prediction accuracy. But to decide on whether the non-linearity of the target PUF is low enough for SVM, it may take some trial-and-error since as we explained, there is no deterministic relationship between the capability of SVM for modeling and the level of non-linearity in the structure of the PUF. In such case therefore, we propose using MLP in general for any PUF with a non-linear structure. We disregard RF models in this work since RFs can in turn be computationally expensive solutions compared to optimal MLP models. While MLP models themselves have the potential to be resource hungry, they instead have modifiable topology from various aspects which opens them to further optimization, while in RF models the open optimization aspects are limited (e.g., number of DTs to include in the model). There are other ML modeling solutions as well, such as Evolution Strategy (ES), K nearest neighbor (KNN), and Quadratic Discriminant Analysis (QDA) that exist in the literature and have been used to model PUF. However, we suffice our explanation with just the former methods since they have been used in abundant for PUF modeling compared to the mentioned ones.

2.9 Using PUF Simulation to Evaluate Real PUF

We mentioned that our focus in this research is set on strong PUFs and using ML methods to manage their CRP enrollment. Of course, as part of the work, we need to perform evaluations coming from PUF data. However, implementing PUF physically can take a lot of effort, especially because during the implementation, the engineers should carefully do the floor-planning in a way that variations between PUF instances in terms of the CRP is only due to the micro-variations. To do this, several works have proposed peripheral solutions that modify the structure in a way that assures the main structural parameters, such as the length of connections between two internal components, are of the same size. Majzoobi et al. in [43] proposes using programmable delay lines as an example of that.

An easier way to assess PUF is to use simulators. Although simulation naturally trims some of the real-world factors, however, the main point which is to assign micro process variations can be handled there since there is visibility and access to all the constituent components in a PUF structure. Up now, there has been several works that based their assessment on PUF simulation. For instance, the work of Ruhrmair in [39], performs assessments on data captured from Arbiter PUF and XOR Arbiter PUF simulators.

In this work we also use PUF simulators due to two major facts. One that is the ease of access to the internal components which gives us the capability of modifying the structure to create new variations of PUF. For instance, we are using the base Arbiter PUF simulator developed by Ruhrmair and used in [39]. This base simulator can be used to create other variants of Arbiter PUF, such as feedforward and double arbiter PUF. The XOR Arbiter PUF is already implemented in the simulator. The second reason is the need to generate large number of instances, which then satisfies a realistic scenario for enrollment. At this point we can see the benefits of using simulator especially compared to works that used real PUF data for assessment, which could only present few numbers of instances due to the cost of implementing the PUF on large number of devices. It is nonetheless essential to assure that the main PUF characteristics are acceptable, such as the randomness, and uniqueness. In our work as we show later, we analyze these metrics in the simulated, and so far the characteristic of PUF instances generated from the Ruhrmair simulator has been acceptable and represent the characteristics of a good PUF.

3

PUF Modeling and CRP Data-base Management

In this chapter, we focus on the provision of a predictive model of PUF during enrollment. Four of our published works [44]–[47] regarding PUF modeling will be explained here. First, we go through the methodology that we built on enrolling strong PUF using machine learning. Then we recall the challenges with the conventional way of providing the model of PUF, such as large number of CRP, prediction error, size of model. We present two optimized modeling techniques we implemented specifically for modeling strong PUF, which are transfer learning and sub-space modeling. We explain their structure and how they work, and we present experimental result as proof of concept.

3.1 Introduction

Due to the large CRP space, strong PUF are potential sources to generate single-use device identifiers or encryption keys. For such applications, it is commonly imagined to either employ a single strong PUF and expect to extract each time a vector of responses or employ multiple strong PUFs and thus a multi-bit output of responses. Such PUF based sketches are useful for device authentication or lightweight encryption key generation. Given also that the CRP space of the constituting PUFs is very large, the system employing such sketch for key generation practically never runs out of single-use encryption keys. Although this is a specific use-case for strong PUF, it shows clearly that the potential exists with harnessing the large CRP space, hence our proposal to use a

predictive model of PUF during enrollment which provides access to the entire CRP space of the PUF.

As discussed before in Chapter 1 section 2.7 and 2.8, there already exists a matured research field for strong PUF modeling, which contains a large number of different techniques that tackle the PUF modeling in different ways. Although these methods are mainly proposed to notify the designers of existing attack methods which aim at modeling the PUF, the benefit of ML-based modeling of PUF, as we discussed earlier, is not limited to that use-case. Alternatively, modeling of strong PUF can be utilized by designers for device enrollment. This use-case has already been discussed in several works such as the Slender PUF authentication protocol in [48], a mutual lightweight authentication method proposed in [49], and an encryption protocol proposed in [50].

Modeling strong PUF, however, can in turn be seen as a heavy task, requiring a large amount of CRP and PUF data to yield an accurate model. Depending on the level of PUF complexity, modeling PUF in turn can lead to requiring over a million CRPs. While capturing a large number of CRPs for one device is not an issue, practicing it for a large group of devices can be seen as a cost issue. The lateral is the setting that designers can face if they do not consider a cost-aware enrollment process.

The primary work that we conduct in this regard is to put together a method for PUF enrollment using machine learning, with attention to cost parameters that measure mainly the amount data needed for training, the time needed for a successful enrollment, the storage space needed to store the model of PUF, and the computation power required to handle the training process. We discuss here that due to the empirical nature of machine learning in yielding a model with desired characteristic, it is required that we learn about the subject of modeling and find the optimal hyper-parameters for training first. Then we adjust a set of so-called control parameters which control the training to perform in a fixed time and data frame during the main enrollment. Later in this chapter, we introduce two optimization techniques for modeling, one which aims at optimizing the training which we call transfer learning, and the other is sub-space modeling, a structural modification on the PUF which allows capturing data from the internal components of the PUF and allow modeling the PUF using a divide-and-conquer approach. Both of the solutions in turn should enable us, as we show by experimental results as well, to model PUF with reduced cost in terms of time and training dataset size.

3.2 Strong PUF Enrollment using Machine Learning: A Methodical Approach

The goal of strong PUF enrollment with ML-based modeling is to replace the conventional CRP database with an estimated model of the PUF. Let us denote the estimated model of PUF as h_{PUF}. The enrollment of PUF with ML-based modeling means that the verifier server will own h_{PUF}, which provides access to the full CRP space of the PUF circuit with some miss-prediction error that is tolerable (see Figure **??**a). This should in turn mean that the h_{PUF} and the PUF circuit itself should respond similarly to any given challenge from the CRP space. Let us consider c_i as a challenge input to the PUF circuit. If we observe the PUF circuit as a function f_{PUF} of c_i, then its estimation can be defined as a function g_{PUF} of c_i and a set of internal values θ of the model. Thus, $h_{PUF} = \{g_{PUF}, \theta\}$. The estimation should then follow (3.1):

$$f_{PUF}(c_i) = r_i \approx r'_i = g_{PUF}(c_i, \theta) = h_{PUF}(c_i) \tag{3.1}$$

where r_i is the PUF circuit's response to the challenge c_i and r'_i is the estimated model's prediction of r_i for c_i. The model then goes through an iterative training phase, where a learning algorithm modifies the internal values with respect to the CRP set and the function g_{PUF}. At the beginning of the training phase, model h_{PUF} has a significant probability of erroneous estimation of the PUF's CRP characteristic. Therefore, the training runs iteratively until the probability of erroneous estimation is converged to zero or an acceptable minimum value.

Since modeling here is done for the enrollment, we define metrics that are important for the enrollment, and we use them to evaluate the cost of training and the performance of the estimated models:

- **Prediction Accuracy** (ϵ)**:** Proportion of correctly predicted responses to total number of predictions.

- **Enrollment CRP Set Size** (css): Size (in bytes) of the CRP set collected to enroll a given PUF circuit.

- **Total Time of Training** (T)**:** The time of training in seconds, up to a point when an estimated model is generated with acceptable ϵ.

- **Estimated Model Size** (ms)**:** A measure of size (in bytes) of the internal trainable parameters θ of an estimated model of a PUF.

Training in Machine Learning is an iterative process where numerous training data-samples are fed to a prediction model and the output of the prediction model is assessed and the result of that is fed back to the model in order to optimize it to increase its prediction accuracy. The PUF enrollment procedure is done by an authorized party with open access to the PUF circuits to collect arbitrary number of CRPs. We refer to this authorized party as the designer.

Before the enrollment process, we assume that the designer first has performed a CRP-readout on a group of silicon Chips. During the CRP readout, the designer captures an initial set of CRPs for each strong PUF circuit. We refer to this initial set as CRPt. We assume that the size of the CRPt is fixed for all the PUF circuits in the same group. The CRPt is then divided into three subsets. We define a CRPtr subset to be used for training the estimated model. CRPval is a subset to evaluate the estimated model during training, and CRPte a subset to evaluate the estimated model after training. We assume CRPte to be considerably larger than CRPval.

During the process of training the estimated models for the PUF circuits, we take the following considerations as well:

- The PUF circuit and its estimated model respond similarly to any randomly given challenge with high probability. In this case, we say that the estimated model has a high value for ϵ.

- The number of CRPs needed to train each estimated model is enumerable and feasible to collect.

- The training process is finite and the training time T for obtaining an estimated model is minimum.

- The estimated model's internal parameter set θ is enumerable and feasible for storage.

- Each estimated model characterizes only its corresponding PUF circuit and has no correlation with other PUF circuits in the same group of PUFs.

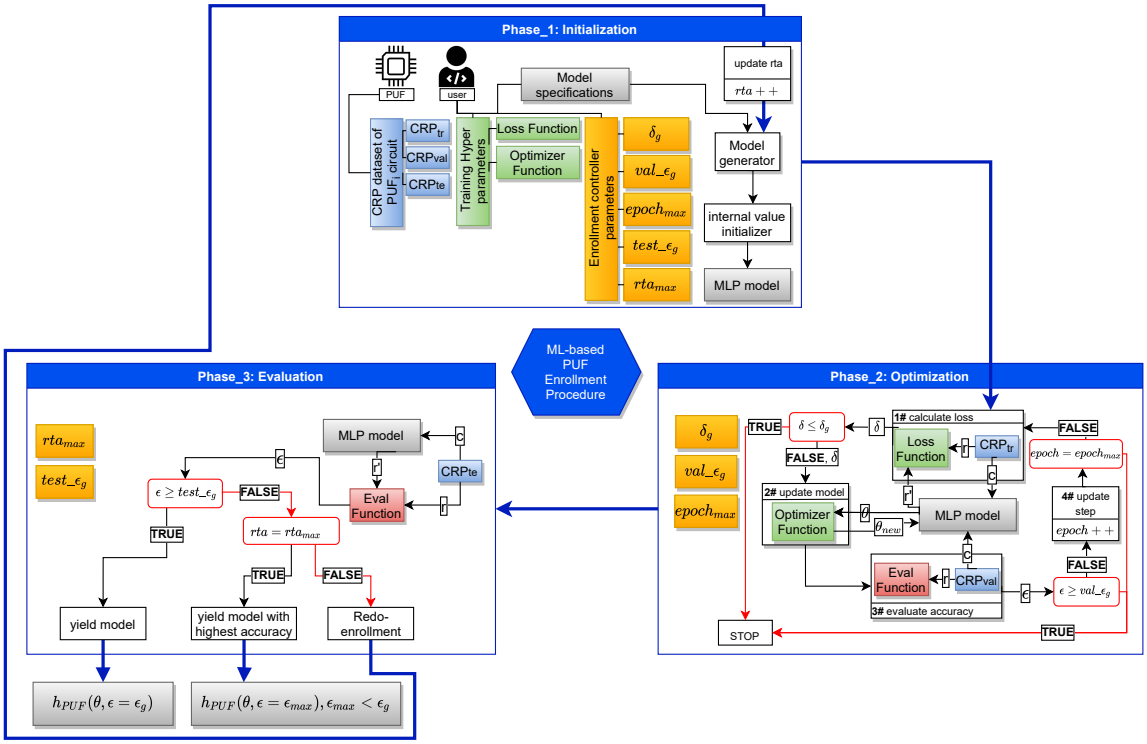


Figure 3.1: Our proposed ML-based enrollment procedure

The procedure of ML-based enrollment can then be developed as shown in Figure 3.1. This procedure will run in three phases to enroll a given PUF circuit: (1) the Initialization phase, (2) the Optimization phase, and (3) the Evaluation phase, which we will explain more in details in the following:

3.2.1 Initialization Phase

Here, we define and initialize the necessary parameters for the optimization and the evaluation phase. The control parameters as described in Table 1 are initialized by designer specified values. These parameters define the target accuracy of the estimated model as well as the maximum period of time the enrollment can take to reach the model with the target prediction accuracy. The Training Hyper-parameters are also initialized in this stage. These parameters are machine learning specific and necessary to be assigned according to the context. The choice of the values for these parameters is up to the designer as well, and the best practice is to inherit the values of the successful practices in the literature that have done the strong PUF modeling. The training CRP set CRPt is also given at this stage, including its three subsets as explained earlier. Finally,

Table 3.1: Control parameters in the enrollment procedure.

Parameter	**Description**
val_ϵ_g	Desired value for ϵ wrt. CRP_{val} subset.
$test_\epsilon_g$	Desired value for ϵ wrt. CRP_{te} subset.
δ_g	Minimum error of mis-prediction wrt. the CRP_{tr} subset.
css_{tr}	Size (in bytes) of the CRP_{tr} subset.
$epoch_{max}$	Maximum allowed training epochs
rta_{max}	Maximum allowed re-training attempts

the estimated model is created in this stage. The model specifications are also given by the designer. After the model is created, it is sent to an initialization process where its internal values are randomized. The model initialization part of the phase can be iterative. It would depend on how the next two phases will perform, which we explain in the following.

3.2.2 Optimization Phase

The training of the estimated model is done in this phase. It comprises multiple functions for training and evaluation of the estimated model. At the beginning of the phase, the estimated model is given a set of challenges from the CRP_{tr} dataset. The model then predicts the corresponding responses. The loss function will take the predicted responses and the actual responses from the CRP_{tr} and compute the prediction loss δ. Then, the loss value δ is given to the optimizer function which propagates adjustments to the internal parameters θ of the estimated model. Then, the Eval function computes the prediction accuracy ϵ of the updated estimated model, using the CRP_{val} dataset. This entire process is called an epoch, which is counted as $epoch$ in the procedure. The procedure undergoes several epochs of training until the desired value of loss δ_g or prediction accuracy val_ϵ_g is observed or $epoch$ reaches max value $epoch_{max}$.

3.2.3 Evaluation Phase

The evaluation phase performs the final prediction accuracy assessment of the updated estimated model over the CRPte dataset. Here, the same eval function measures the prediction accuracy of the model over CRPte. This evaluation in turn tries to emulate the scenario where the estimated model is invoked during mission mode to communicate with the PUF circuit. In such case, it is justified to have the CRPte set size be considerably

larger than that of CRPval. The control sequence in this phase compares the prediction accuracy ϵ of the model with the $test_\epsilon g$. If greater, then the enrollment yields the model as an estimated model with desired accuracy for enrollment. If the prediction value is less than $test_\epsilon g$, however, the control sequence sets to redo the training from the initialization phase where the model's internal parameters are initialized. The number of re-training times is also counted in the procedure with rta counter. If rta reaches $rtamax$ as defined by the designer, then the control sequence yields the enrollment with the model with maximum accuracy ϵmax from the previous training attempts, where $\epsilon max < test_\epsilon g$.

3.2.4 Methodology

Recall that we assume in this work that enrollment is done for a very large number of PUF circuits. Primarily, the user has to initialize the control parameters before to conduct the enrollment procedure. We know already that the ML-based enrollment procedure is empirical, meaning that there is no deterministic parameter initialization known beforehand to initiate the ML-based enrollment with and yield the desired results. Instead, the optimal values to start with are empirically drawn from a test case. Accordingly, for enrollment of PUF using ML-based modeling, we suggest that the user performs the enrollment in two parts:

- Part1: Set arbitrary values for the hyper parameters and the control parameters $val_\epsilon g$, $test_\epsilon g$ and δg, epochmax and rtamax. In addition, define the minimum and maximum bound of css. Then, perform the enrollment procedure over a discrete range of css values (arbitrary selections between the min, max bound) separately and evaluate at which css the desired ϵ is obtained with acceptable T.

- Part 2: Update the control parameters with implications of the optimal values for ϵ, T and css, obtained from part 1, and resume the enrollment on the rest of the PUF circuits with the updated control parameters.

Our focus here is mainly on part 1. Thus, our goal is to show how the observations obtained in Part1 can help the user to define the optimal values for the control parameters. Speculatively, to be cost-efficient for the enrollment of a given large group of PUF, the user is able to find a minimum value for css given the implications he receives from the evaluation on the primary subset in part 1. Therefore, he will be able to avoid larger

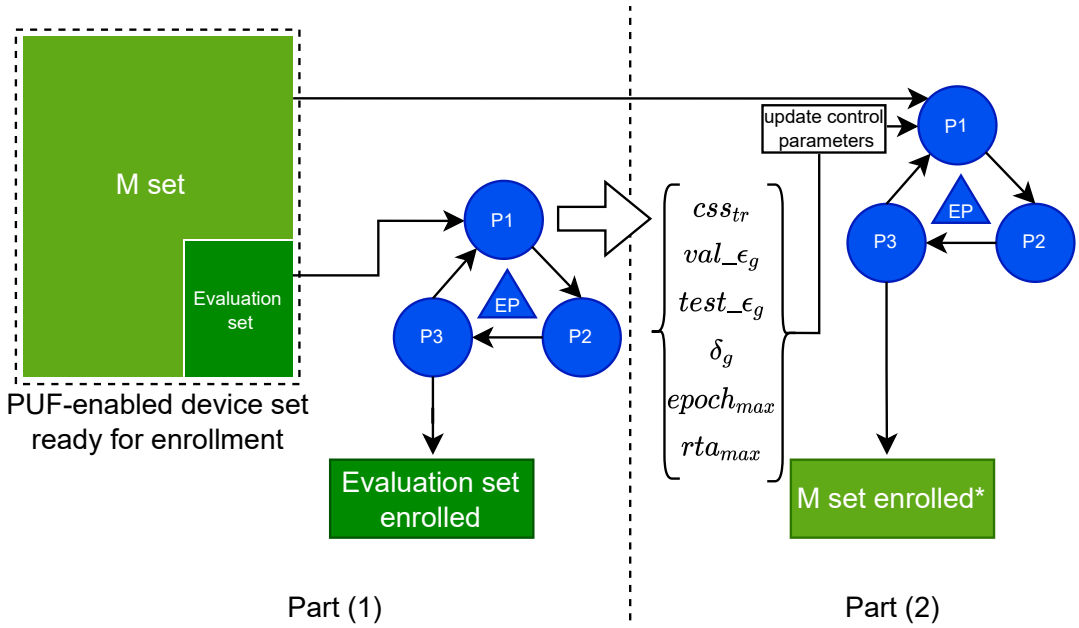


Figure 3.2: Our proposed enrollment method. Here, EP refers to our proposed enrollment procedure. M set also refers to the main set of PUF devices to be enrolled with adapted control parameters.

css and save time during the CRP-readout for the remaining large number of PUF circuits pending for enrollment.

Since it is an empirical method, we should first define the hard bounds for either one of the control parameters. We suggest to do this for the most critical parameter such as $csstr$. After exploring the values within the hard bounds, then find the optimal points for the cost values. For instance, looking at where css or T is minimum, the ϵ is maximum and css is minimum. Accordingly, update the values of the control parameters (see Figure 3.2).

Additionally, an exploration of ms can be done. ms, however, is mostly relying on the parameters which constitute the structure of the probabilistic model, such as number of neurons and weighted connections for each neuron if the model is an Artificial Neural Network (ANN). ms parameters are quite numerous. Therefore, the exploration over ms should be selective, such as exploring different model structures that have already been proposed in the literature. The method of grid searching the hyper-parameters for training also exists, such as the learning-rate value, the optimization function, etc. This can be done on the side of the enrollment method as we define here.

Once the enrollment on the evaluation set is complete after exploring different val-

ues for the control Parameters, there will be two products: one which is the enrollment of the devices in the evaluation set, and second is the optimized control parameters which then can be used in the enrollment of the M set (see Figure 3.2). This will constitute the second part of the method. Contrary to the first part, in the second part, no bound for control parameters are set. Instead, final values for these parameters are given, which are coming from the first part. The outcome of the second part of the method is to accept an hPUF model which meets the qualifications according to ϵ parameters or discard the model which does not have the desired quality. In case of discarding the models, their corresponding PUF will be queried again for more CRP-readout. Nonetheless, we speculate that, with the optimized control parameters exported from part one, the population of discarded models in the second part should be minimized considerably.

3.2.5 Evaluation Work

In this section, we show in an experiment how we analyze css, T and ϵ over a small batch of 100 2-XOR Arbiter PUFs. We considered using XOR Arbiter PUF as our target PUF model family in our evaluation code. We conduct our experiments on data generated from a Python based Arbiter PUF and XOR Arbiter PUF simulation. We elaborate on XOR Arbiter PUF and its simulation in Python in the following.

We reused the XOR Arbiter PUF simulator developed by Ruhrmair as described in [26]. The source code of this simulator can also be found in [51]. In this simulator, the two racing signals' propagation delay is modeled as the sum of the delays in each stage. The delay parameter values in the Python-based implementation of APUF and XOR Arbiter PUF simulator are generated randomly according to a standard normal distribution, with mean 0 and standard deviation 1.

For our experiment, we generated 100 instances of 128-stage 2-XOR Arbiter PUF. We then randomly generated 35,000 challenges for each instance and recorded their corresponding response. Thus, we stored 100 CRP datasets with 35,000 CRPs in each set. Note that the datasets generated from the simulated PUF instances do not simulate the instability that is inevitably present in real PUFs. We intentionally chose the instability-free condition, since the presence of instability is a new fold of complexity that can affect the modeling results, and thus it needs to be discussed thoroughly in a separate set of experimental work.

To assure the reliability of the simulated instances and the generated CRPs, we mea-

Table 3.2: Measurements on 100 PUF instances CRP sets.

Average Randomness	**Maiti's Uniqueness**	**Hori's Uniqueness**	**Average Diffuseness**
0.9419	0.4999	0.9899	0.9972

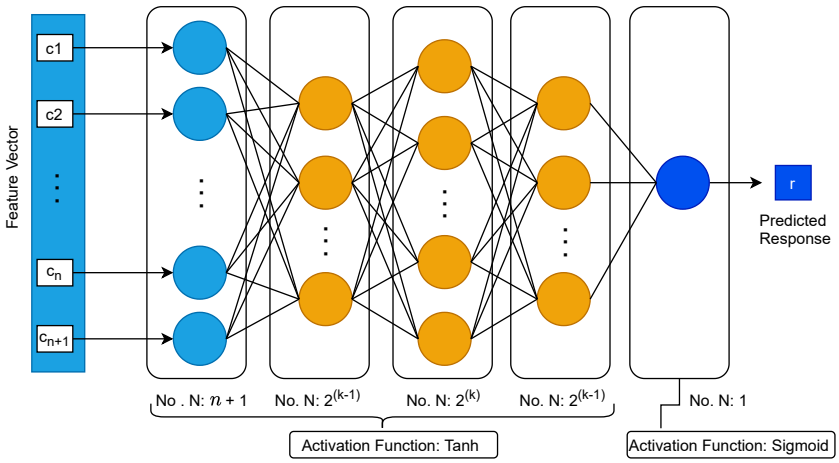


Figure 3.3: Illustrating the MLP structure proposed by Mursi et al. in [6].

sured the randomness, uniqueness, and diffuseness, using the formulations proposed in [52], [53]. A collective use-case of Hori's Uniqueness and Maiti's Uniqueness, as well as the randomness and the diffuseness, can be found in [54]. Our measurements of uniqueness, randomness, and diffuseness over 100 instances of 128-bit 2-XOR Arbiter PUF are brought in Table 3.2. Note that each CRP set considered for these measurements comprises 10,000 CRPs. As shown in the table, the measurements on all the metrics are close to the ideal thresholds for each metric as we explained earlier in Chapter 2, Section 3. Therefore we can consider the instantiated PUFs from the simulator as good PUFs.

We chose our estimated model to be Multi-Layer Perceptron (MLP), which is a variant of Artificial Neural Network (ANN) models. Mursi et al. in [6] has proposed a structural definition of MLP model for modeling XOR Arbiter PUFs that has the potential to converge faster with a considerably lower number of CRPs for training compared to other modeling structures such as ones discussed in [26], [55], which are based on Logistic Regression (LR), and [56], which is based on Artificial Neural Networks (ANN). A schematic of Mursi's proposed MLP structure is given in Figure 3.3. Here, k is with the number of XORs in an n-stage k-XOR Arbiter PUF. The mentioned feature vector in the figure is solely a function of the applied n-bit challenge c. As also described by Ruhrmair in [26], the feature vector as $\vec{\Phi}$ can be defined as $\vec{\Phi}(c) = \prod_{i=1}^{k}(1 - 2b_i)$, where

Table 3.3: Hyper-parameters set for the initialization phase.

Parameter	**Value**	**Parameter**	**Value**
Optimization Function	Adam	Loss Function	BCE
Learning Rate	0.001	Weight Initializer	Taiming Uniform
Bias Initializer	Uniform	* $epoch_max$	400
* $test_\epsilon_g$	90%	* rta_max	10
* val_ϵ_g	99|%	δ_g	0.01
CRP_{{te}} Set Size	20,000		

b_i is the ith bit of the challenge c.

Accordingly, for a 128-stage 2-XOR Arbiter PUF, the MLP model we use in this work has an input layer with 129 neurons, first hidden layer with two neurons, second hidden layer with four, and a third layer with two neurons. The output would also have one classifier neuron. For the training hyper-parameters and the enrollment control parameters, we considered the values given in Table 3.3.

3.2.6 Evaluation Observations

We did an exploration to find the ideal values for csstr and T. The choice of the lower bound of csstr and the upper bound was also arbitrary. We inferred from the previous studies that the lowest value for csstr for a 2-XOR 128-it XOR Arbiter PUF is a value of about 3000 CRPs. Therefore, we choose that as the lower bound. For the upper bound, we chose 10,000 CRPs. We speculated that the characteristic we observe around this number of CRPs can be interpolated for larger sizes of csstr as well.

We used Pytorch in Python 3.7 to build and train our ANNs. We conducted our experiments on a PC running windows 10 with an Intel core i7 8th Gen CPU and 16 GB of memory. We developed our experimental Python codes using Spyder 4.0.1 on Anaconda Navigator 1.9.12.

The results on the performance of training with various training set sizes are reflected in Figure 3.4. Looking at the results, we can identify at what csstr value the probability of reaching the desired ϵ is very low, which is between 3000 to 4000. Note that the model accuracy at these csstr values is ϵmax as we indicated in the procedure. In addition, looking at the range between 4500 CRPs to 10,000 CRPs for csstr, it infers that, with increasing the set size, the probability of reaching the desired ϵ tends to stabilize at a value above the target $test_\epsilon g$ as we defined in the test phase. There exist some outliers that are cases where the desired ϵ could not be reached; therefore, the model

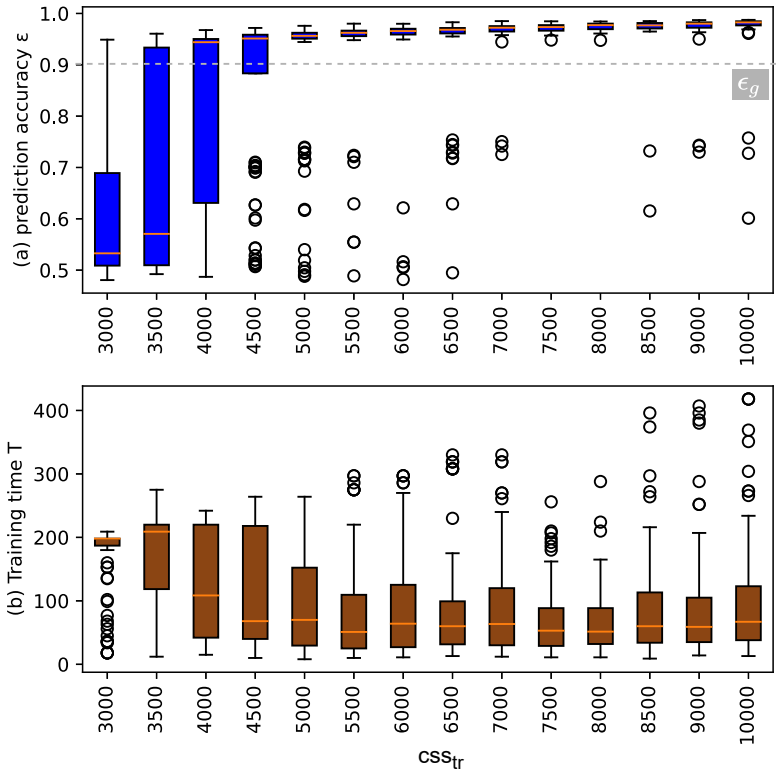


Figure 3.4: Illustrating the distribution of prediction accuracy ϵ and the distribution of total training time T over various csstr.

with ϵmax is yielded.

This can characterize the possibility of having discarded HPUF models during Part 2 of the enrollment method as we explained earlier. Moreover, the training time T tends to decrease as well at the beginning with increasing the set size up to 7500. We imply that this reduction in time of training is due to a significant decrease in re-training attempts until desired model is yielded.

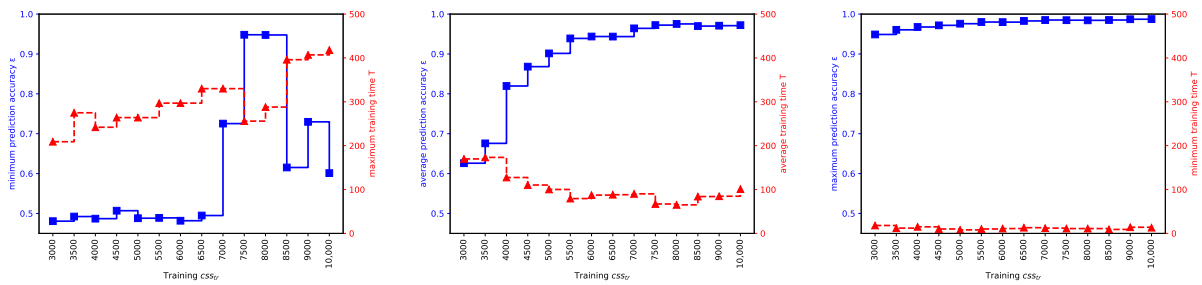
Plots shown in Figure 3.5 also show the minimum, maximum, and average values of T and ϵ. Looking at Figure 6.a, we can observe what are safe values of csstr in terms of delivering the target prediction accuracy ϵ with maximum probability. In this scenario, look for $\epsilon > 0.90$, that is, the values between 7000 to 8000 CRPs. Looking at values of ϵ and T shown in Figure 3.5.c, one can infer at what csstr values there is a chance of obtaining the target ϵ. Here, at 3000 for instance, 3000 CRPs could yield a model with $\epsilon > 0.90$. This, however, means that a designer needs to improve other factors to increase the chance of obtaining model with target ϵ. For instance, by using a better initialization

technique or modifying the model structure, which could in turn affect the model size ms consequently. By also looking at Figure 3.5.b, we can infer at what csstr values we have an increased chance of obtaining the target ϵ. For instance, for csstr $>$ 5000, it is possible to obtain prediction accuracy $\epsilon >$ 0.90. However, for this csstr, there is a considerable chance that models with $\epsilon <$ 0.90 are yielded. Choosing this option depends on the cost of re-querying the corresponding PUFs for the outlier predictive models with low prediction accuracy. If the cost of re-querying is amenable by choosing the low csstr for the majority of the PUF devices for the first query, then it could be considered a potential choice in terms of lowering the overall cost of enrollment.

We could have two approaches here: (1) To choose the minimum css that has a chance to yield some hPUF model with sub-optimal ϵ; (2) To choose css which yields maximum ϵ and has a negligible chance of yielding hPUF with sub-optimal ϵ.

Choosing the first option can yield in overall reducing the CRP read-out cost since we chose the minimum csstr. However, since there is also the chance of yielding hPUF with sub-optimal ϵ, then there may be some additive cost of re-querying the PUFs with discarded hPUF for re-enrollment. If the cost of re-querying is tolerable, then the first option could potentially be the cost-efficient choice.

Choosing the second option, however, could yield the overall increased CRP read-out cost. We also saw that training time T can also be minimized if the prediction accuracy ϵ is maximized with increasing css. However, the chance of yielding hPUF with sub-optimal ϵ is negligible on the other hand. This option could be a potential choice for cost-efficiency if the cost of re-querying the PUF is high.



(**a**) (**b**) (**c**)

Figure 3.5: Illustration showing the minimum, average, and maximum training time T and prediction accuracy ϵ for various cc_{tr}. (**a**) Max T and Min ϵ; (**b**) average T and ϵ; (**c**) Min T and Max ϵ.

We also measured the size of the trained estimated models. The MLP used in this

study comprises 282 trainable parameters in total. Each parameter is 32-bit floating point, yielding in total a model size of 1128 bytes. Note that we do not need to save any metadata regarding the internal connectivity of the MLP, since all layers in the MLP model are Fully Connected. While we cannot make a thorough comparison at this level of evaluating the method, we can at least draw a primary conclusion that storing an estimated model of a strong PUF circuit potentially takes much less space, compared to that of a CRP set. For instance, the training CRP set used here to obtain the most accurate model includes 10,000 CRPs, equal to 161 KB of storage size, whereas the estimated model size is roughly above 1 KB.

The uniqueness of the trained estimated models is also an important characteristic, which means that no estimated model should respond similarly to two different PUFs. Although this is not a metric related to the cost of enrollment, it is, however, essential to ensure that models correspond only to their equivalent PUF circuit. This should satisfy the fifth consideration we discussed earlier in this chapter. We refer to it as measuring the uniqueness of the estimated models. Uniqueness here is observed as the prediction accuracy of each trained estimated model over a given CRP dataset coming from each PUF circuit. Since we have 100 PUF circuits and their corresponding 100 estimated models, we therefore measured 10,000 cases for the uniqueness. We considered two cases, one which is training the models with csstr = 3000, and one with csstr = 8000 CRPs. The results of this measurement are brought in Figure 3.6. We expect for csstr = 3000 that the models have low similarity to their corresponding PUF. Looking at Figure3.6.a, we see that only a selective number of cases have ϵ above 0.70. Nonetheless, for all cases, it is apparent that no similarity with ϵ higher than 0.6 is achieved. We also observe on Figure 3.6.b that estimated models trained with csstr = 8000 show similarity uniquely only to their corresponding PUF circuit with high accuracy. We cannot infer directly from these observations that, for any trained model on a PUF circuit in general, the model shows CRP similarity only to the corresponding PUF. Since we suspect that there might exist some PUF designs that intentionally have CRP similarities scattered between various PUF devices, we can nonetheless infer that, for a set of PUF devices for which their PUF characteristic shows good random and unique behavior, predictive models trained using enough CRPs from a PUF represent uniquely that PUF only, while having no similarity to other PUF instances.

3.2.7 Applicability in the Related Works

The methodology we developed here can be used in the protocols that demand a model of in their authentication and encryption key generation process. Below, we discuss several protocols as such that exist in the literature, and explain briefly why providing a model of PUF is an essenial part of them.

The Slender PUF protocol proposed by Majzoobi et al. in [48] is an authentication protocol that uses strong PUF. In this protocol, the authors propose a substring matching mechanism, wherein the substring in the protocol is a sliced vector from a vector of generated responses both on the PUF device and the verifier server. Authors in this work assume that the verifier server has access to a compact model of the PUF, which is able to generate a response for any given challenge vector, similar to that in the original PUF circuit. The sliced vector of responses is sent from the PUF device to the verifier, and convoluted over the entire response vector that has been generated on the verifier server, using the compact model of the PUF. The authentication is successful once the sliced vector shows maximum correlation to a subpart of the verifier's response vector. Since, in this protocol, the sliced response vector is exchanged on a public channel, it is obliged that, once the vector is used, it is discarded and never used again to prevent replay attacks. This in turn requires that the verifier server has access to a very large amount of the CRP space of the PUF, which is in turn guaranteed by using the model of the PUF. Thus, the model of the PUF should correspond as accurately as possible to the PUF circuit in order to suffice the requirement.

Another novel mutual authentication protocol has been proposed by Idriss et al. in [49], [57], which is based on a challenge-challenge communication mechanism between the PUF-enabled device and the verifier server. It is also assumed in this work that the verifier server acquires an accurate model of the PUF for enrollment. During the mutual authentication, after exchanging the device IDs, the PUF-enabled device generates several random CRPs. To authenticate the server, the challenge vectors of the generated CRPs are sent from the device to the verifier server. On the server side, for each received challenge vector ci, two new random challenge vectors cj and ck are generated, such that the XOR of the response values corresponding to each of the two newly generated challenge is equal to the response of the received challenge vector. In other words, the $A(cj)$ XOR $A(ck) = r' == r = A(ci)$ should hold true, where A is the model of the PUF circuit. Once the cj and ck are generated for every ci received on the verifier server, they are sent to the PUF-enabled device. On the PUF-enabled device, it is then checked

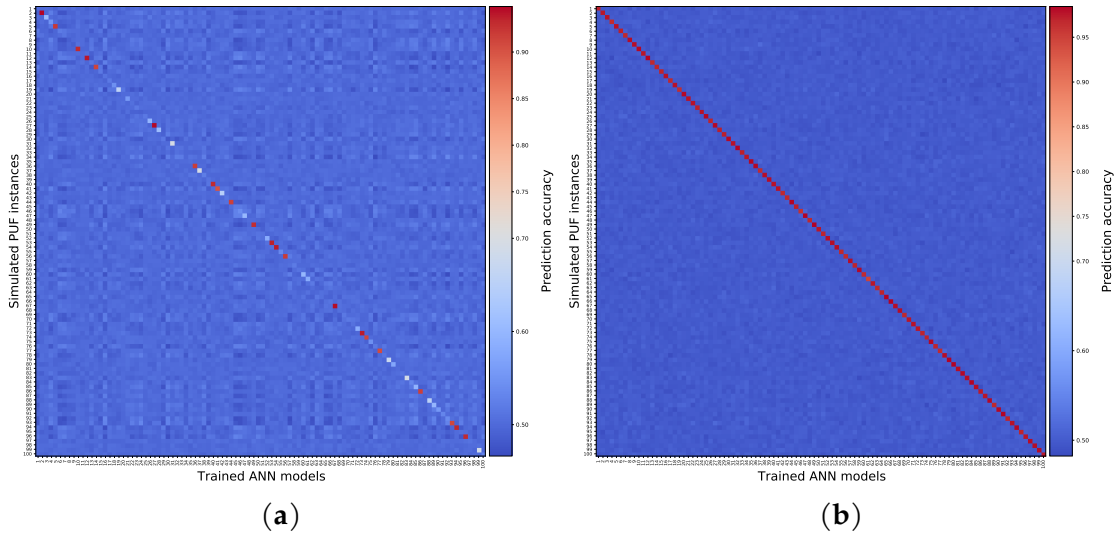


Figure 3.6: Similarity matrices showing the similarity between 100 trained models and 100 instances of a 128-stage 2-XOR Arbiter PUF. (**a**) css_{tr} = 3000 ; (**b**) css_{tr} = 8000.

to see if the equation PUF(cj) XOR PUF(ck) = r′ == r = PUF(ci) is true for the majority of the received challenge vector pairs. Once the device authenticates the server, it sends a new challenge vector pair set (the same as the verifier did) to the verifier server for authenticating the device. Since this mutual authentication method is based on random generation of the challenge values for every authentication request, it is assumed that the verifier server has access to a large CRP space where, for every randomly generated challenge vector, a response value can be provided. This of course is guaranteed by using the equivalent model of the PUF, which is highly accurately trained.

A PUF based key generation protocol has been proposed by Quadir et al. in [50], which uses a machine learning generated predictive model of the PUF on the TTP server for mutual key generation. Here, the authors propose mutual key generation by exchanging only a serial number, which in turn is the challenge to the PUF device and the predictive model, respectively. It is expected of course that both the model and the PUF device generate the same response value. The response value of course is prone to variations due to device instability and model miss-prediction, which is why the authors also propose using helper data and error correction codes to recover the original key generated on the TTP server for the PUF device. Similar to the idea of PUF authenticating in [57], here only the challenge values are exchanged and no responses, in order to avoid model-building attacks. The protocol proposed here also refreshes the key after a certain period. This feature of course needs both the device and server to have access

to a large CRP space, which is again for the verifier server, provided using the model of the PUF. This way, the users will be able to refresh keys frequently and each time guarantee that a new value is generated.

The advent of such protocols for device authentication and key generation enables a secure implementation of One-Time-Password (OTP) methods to be more feasible and reliable than before. Since the CRP space of strong PUF is considerably large, it can be easily guaranteed that every newly generated password is unique. On the other hand, the reliability of the key is of importance, which is partially assured by providing a highly accurate model of the PUF. Additionally, once these protocols emerge into a large spectrum of embedded systems and connected devices, it is expected that the enrollment process now relying on machine learning-based modeling can be practiced for a very large volume of devices. Therefore, the cost of enrollment, as we defined its constituent terms in this work, finds their importance and needs to be managed properly.

In the following section we will now elaborate on transfer learning as an optimization technique in modeling PUF. TL is a practice in ML that is trending nowadays due to its potential. The point about TL is the reusability of trained data into new subjects of modeling. In the coming chapter, we elaborate on how transfer learning is applied in modeling strong PUF, in specific the family of XOR Arbiter PUF. We then propose using TL with a known MLP model that is tailored for XOR Arbiter PUF modeling. Practicing TL on MLP in modeling PUF is a new approach and we show here that This practice shows better results in terms of training CRP dataset size compared to an already practiced TL on CNN models.

3.3 Transfer Learning: A Modeling Technique to Reduce Training Data

The success in modeling strong PUF depends on the number of CRPs given to train the estimated model. It is proven that the minimum number of CRPs required to yield an accurate model is proportional to the level of complexity in the structure of the PUF [58]. Thus modeling a strong PUF can be very data demanding for PUF which are largely complex [58]. For instance, modeling k-XOR Arbiter PUF with k >= 5 need above 1 million CRPs to yield estimation accuracy above 90%.

While the increased complexity in the structure of strong PUF seems a workable solution against model-building attacks, new research such as the ones found in [6], [56] try to tackle PUF complexity by proposing novel probabilistic modeling based on deep learning methods which are structurally tuned to converge faster and require lesser data for training. Moreover, a recent work suggests that a transferring trained data from one PUF to model a new PUF can yield accurate model with reduced number of CRPs required for training [59]. This work proposes using Transfer Learning on CNN models which are relatively more complex models than MLPs as used in [6].

One of the ideas we wanted venture into was to model strong PUF with MLPs using Transfer Learning to reduce the training CRPs. Our starting point was set to use the latest MLP proposed by Mursi et al in [6]. We set to experimentally prove that it is possible to initialize some of the dense layer weights of a given MLP model with pre-trained values, and consequently train the model accurately with a reduced CRP dataset. In the following, it will be demonstrated that our modeling technique was able model strong PUF with less resources compared to its predecessor techniques [6] and [59]. What we tried to deliver was the following:

- Schematic of a modeling procedure of strong PUF using MLP and Transfer Learning.
- Evaluation with simulated noise-free data of several variants of strong PUF models to show that reusing dense layer weight values can further decrease the required number of CRPs for training compared to [6].
- Experimental assessment with simulated noisy data to show the resilience of modeling scheme based on Transfer Learning to PUF instability.

- A comparison between our proposed Transfer Learning method and wang's Transfer Learning on CNNs in [59] in modeling some variants of strong PUF.

3.3.1 How Transfer Learning Works

Depending on how the internal values of an ANN are initialized, the training process can lead faster or slower to a point of convergence. Several techniques for ANN weight and bias initialization have been proposed. A selection of these techniques is discussed in [60]. In contrast to these solutions, another solution called Transfer Learning exists which is based on reusing the values of internal parameters of the trained estimated model [61]. It is proven that Transfer Learning is a potential technique to mitigate the large data dependency issue in deep learning. This addresses the demand on very large number of samples in training in order to yield an optimal prediction accuracy, which also appears in modeling variants of strong PUFs.

In the context of strong PUF modeling, the main goal is to reduce the required CRP in building an accurate estimation model for a PUF circuit *PUF GOAL*. Here Transfer Learning suggests extracting θ from an already trained model, we refer to as *MODEL P* which is an estimation of a PUF circuit we refer to as *Prime PUF*. A Transfer Learning Plan then decides which part of θ can be reused. Based on the Transfer Learning Plan, θ' is generated from θ. The modified θ' is then assigned accordingly to the internal parameters of a new model *MODEL G*. *MODEL G* is then passed to the training process for estimating *PUF GOAL*. In the following we elaborate on our proposed Transfer Learning Plan which can be applied to MLPs in modeling strong PUF.

3.3.2 Proposed Method

MLPs have been a common selection as a fit probabilistic model structure to estimate large and complex strong PUFs [6], [54], [56]. Although other neural network models exist such CNN that have the potential to model strong PUFs with large complexity. However, CNN's computation overhead exceeds that of an MLP with the same capability. Therefore, it is interesting to see if the potential of Transfer Learning can emerge with MLPs in modeling strong PUFs. This has not been practiced before. At the time, the latest practice of Transfer Learning for modeling strong PUF we found is in [59] which uses CNN models. We continue in the following to elaborate on how we can successfully use Transfer Learning on MLP models in modeling strong PUF to reduce

the number of CRPs required for training. In this work we take the following assumptions:

- A target PUF exists which is the goal to model. We call this the Goal PUF.
- Access to the Goal PUF is limited. Thus only a few CRPs can be collected.
- The structure of the Goal PUF is known to the adversary.
- A PUF circuit is available with full control. We call this the Prime PUF.
- Access to the Prime PUF is unlimited. Therefore, as many CRPs as required can be collected.
- The structure of the target PUF is known and is similar to the structure of the Prime PUF.

In common Transfer Learning practices, it is suggested to transfer the weight values from the initial layers of an ANN model. It is assumed that the primary features of the target domain are learned in the primary layers of an ANN. In this work, we observed in a preliminary experiment that if we transfer the first layer's weight and bias values, the training performance is negatively affected. Meaning that it will take more CRPs and time of training until convergence. On the other hand, we observed that transferring the weight and bias values of the hidden layers have an opposite effect. Meaning that the training improved in terms of number of CRPs required for training, as well as the success rate of the training and the time of training until convergence.

In using the lateral approach, we measured the average distance between each corresponding layer's weight and bias values of Model P and Model G. We observed that the distance between corresponding weight values on Model P and Model G on the first layers, are considerably higher than that in the subsequent hidden layers. Accordingly, we speculated that the primitive features learned in the first dense layer of Model P are device specific. Therefore, transferring these values to Model G can lead to the possibility of requiring more CRPs and time of training. On the other hand, the weight values in the hidden layers and the output layer of a trained model in general, could correspond to learned features in higher levels of abstraction regarding the characteristic of the target PUF. Therefore, we speculate that the weight value in the hidden layers have potential to be reused in modeling strong PUF with the same structural complexity.

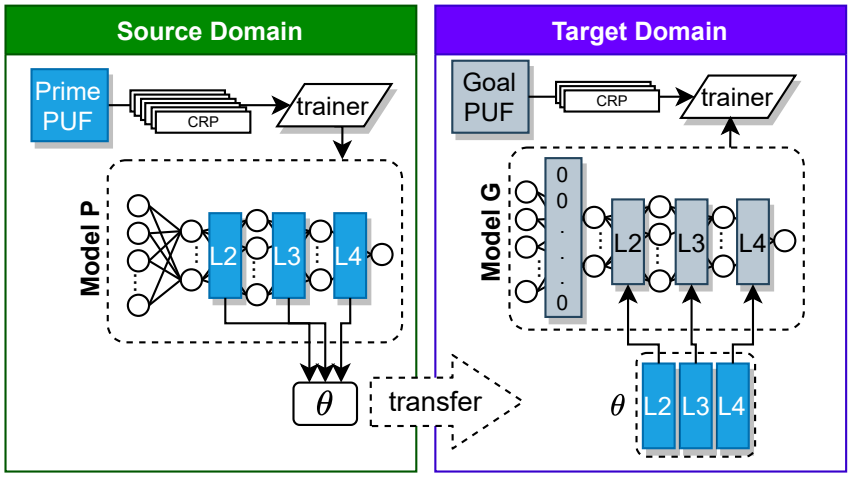


Figure 3.7: Our proposed Transfer Learning Plan, based on reusing the hidden layers of Mursi's MLP model.

Accordingly, we make a Transfer Learning plan to reuse the values of all hidden layers' weights and biases, excluding the weights between the input layer and the first hidden layer. A schematic of our Transfer Learning plan is shown in Figure 3.7. We take the following steps to perform our modeling method:

- Step 1) Initialize an MLP (Model P) to model the prime PUF circuit.
- Step 2) Train Model P with the CRPs captured from the prime PUF instance.
- Step 3) Initialize another MLP model (Model G) with the similar structure to the prime model. All weights of Model G are initialized with zeros.
- Step 4) Overwrite the weight and bias values of layer 2 to layer l of Model G with the weight and bias values of layer 2 to layer l of Model P. Here l is number of layers in the model.
- Step 5) Train Model G with a CRP set captured from the PUF goal circuit.

Our evaluation in this work is based on modeling variants of XOR Arbiter PUFs using simulated data. Here we first elaborate on the structure of XOR Arbiter PUF, and then the simulation code which we used to generate our CRP dataset for modeling. We will then elaborate on our model training setup and discuss our experimental results.

Table 3.4: Training specifications & Hyper-parameters

Parameter	**Value**
Optimization function	Adam
Loss function	BCEloss
Learning rate	0.001, 0.0001
Weight initializer	Kaiming Uniform
Bias initializer	Unifrom
Maximum epoch	2000
Maximum re-training attempts for Transfer Learning disabled	10
Maximum re-training attempts for Transfer Learning enabled	2
Training batch size	= Training set size
Optimal Test Accuracy	90%

3.3.3 Experimental Setup

We reused the Python-based XOR Arbiter PUF simulator developed by Ruhrmair and described in [39]. Again, the code of this simulator is available in [51]. Using the python PUF simulator, we generated 10 instances of 64, 128-stage 2, 3, 4-XOR Arbiter PUF variants. Noting that this simulation does not inherently include the PUF instability noise. To simulate noisy PUF, we randomly selected CRPs from each dataset and flipped the response. Accordingly, we generated noisy CRP datasets for each PUF variant, with 2%, 5% and 10% noisy CRPs in each dataset.

On the modeling part, we recreated Mursi's ANN in [6]. This model is an MLP with 3 hidden layers. The number of neurons in each layer are 2(k-1), 2k, 2(k-1), for the first, second and third hidden layer, respectively. Here k associates with the number of XORs in a n-stage k-XOR Arbiter PUF. The activation function used for all layers except the output layer is hyperbolic tangent function (Tanh). The output layer uses the Sigmoid activation function.

In terms of the training specifications and hyper parameters, our entire parametric consideration for the training are given in Table **??**. We implemented the model creator and the trainer code with Pytorch on Python 3.8 with Anaconda IDE and the Spyder editor on Windows 10. The system we used for training has 2 Nvidia Quadro RTX 5000 graphics cards. Each card has 8 GBs of dedicated memory, 3072 cuda cores with 448 GB/s memory bandwidth. The system has also 2 intel Xeon silver CPUs with 2.2 GHz speed, 128 GBs of RAM, and 10 TB of storage. It is worth noting that during our exper-

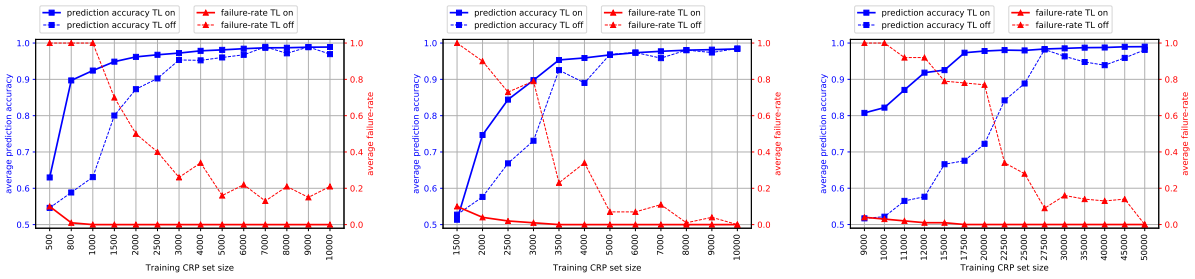
iments, only one of the Quadro RTX 5000 was enough. Also since the dedicated GPU memory was large, we considered the training batch size to be the size of the training set, and also increased the number of epochs accordingly. We clarify that the maximum epoch indicated in Table. **??** should not lead to and overfitting problem, since the fitting iteration (updating the trainable parameters θ) is performed only once per epoch (due to the batch size being the same as training set size). This is similar to having batch size $= \frac{training\ set\ size}{10}$, and maximum epoch $= 200$. However, the larger batch-size allowed us to get a better training performance on GPU.

3.3.4 Experimental Results

In modeling each variant of XOR PUFs for various training set sizes, we measure the following: Averaged prediction accuracy $= (\sum_{i=1}^{10} \epsilon_i)/10$ Averaged failure-rate $= \frac{\sum_{i=1}^{10}(NTr_i)}{MAX_N Tr}$ Where ϵ_i and NTr_a, indicates the trained model's prediction accuracy and number of retraining attempts, respectively., of modeling the ith PUF instance of a given PUF variant and $i \in \{1, 2, \ldots, 10\}$. MAX_NTr also indicates the maximum re-training attempts as given in Table. **??**.

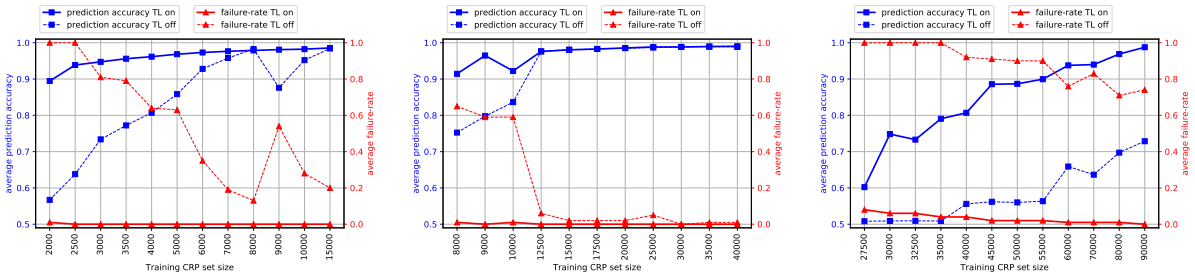
We first compare our method to Mursi's MLP with random initializer. Figure 3.8 shows the training results for the variants of 64-bit XOR PUFs. The first observation is on the average prediction accuracy. If we consider 90% as the target accuracy, we see that training with our Transfer Learning Plan leads to the target accuracy with fewer CRPs in all cases. The average failure-rate also is almost zero using our method. We measured the same for variants of 128-bit XOR Arbiter PUF as shown in Figure 3.9. The same results as that in modeling 64-XOR PUF variants can be seen here. Where the target accuracy above 90% can be achieved with fewer CRPs using our method for initialization. The average failure-rate also is always less than modeling with random initializer.

Our Transfer Learning plan also shows to have a lead against modeling with random initializer where the training data includes some noisy CRPs. Figure 3.10 shows the results of training the 64-bit 4-XOR Arbiter PUF variant in different levels of noise. It is apparent that the difference between the deterioration rate in modeling with random initializer and our proposed method is significant. For instance, in the case of 10% noise and for the maximum number of CRPs given for training, the baseline's prediction error has increased by almost 20%, whereas the case with Transfer Learning, it has dropped



(**a**) 2-XOR (**b**) 3-XOR (**c**) 4-XOR

Figure 3.8: Modeling variants of 64-bit XOR Arbiter PUF with and without Transfer Learning.



(**a**) 2-XOR (**b**) 3-XOR (**c**) 4-XOR

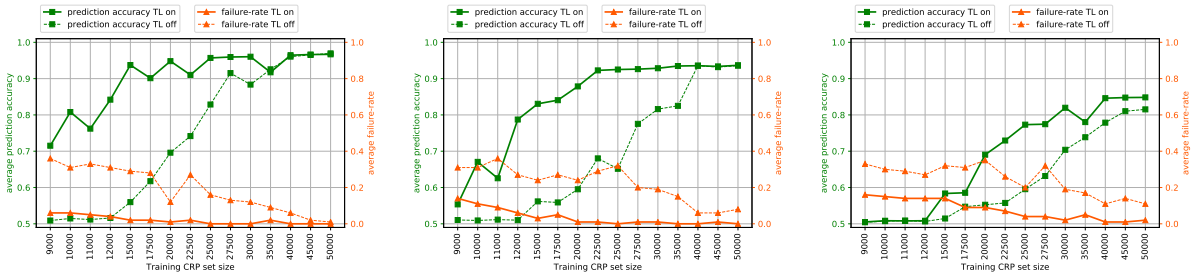
Figure 3.9: Modeling variants of 128-bit XOR Arbiter PUF with and without Transfer Learning.

for 15%. Although it is apparent also that the failure-rate has increased for both cases. Yet the difference of the average failure-rate between the case of modeling with random initializer and our proposed method is significant. Same conclusions can be drawn in modeling a variant with larger challenge space. Figure 3.11 shows the results of training the 128-bit 4-XOR Arbiter PUF with different levels of noise. The results prove that the modeling with our Transfer Learning plan yields the similar advantage.

The observation on modeling with noisy data suggests that using Transfer Learning for initialization has the potential of more resilience to noise compared to a baseline where initialization is performed randomly given any training CRP set size. We also compare our method to the Transfer Learning method proposed by Wang et al. in [59]. In their work, Transfer Learning is done on CNN models to reduce the required number of CRPs for training XOR Arbiter PUF variants. Their method relies on transferring the convolutional layer's weight values from the source domain to the target domain. This means that their method is applicable on CNNs only. Since their CNN specification

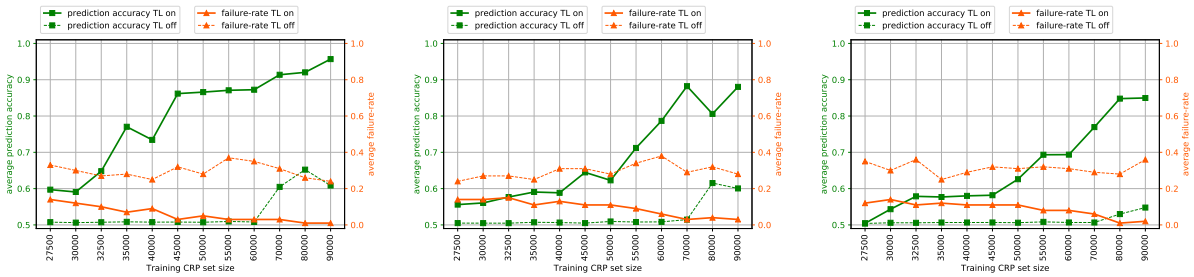
was not given, we could not reproduce their modeling technique, as we did for Mursi's MLP in [6]. However, their work is the only attempt of modeling PUF with Transfer Learning, and closest to our work.

To compare, we consider the modeling targets (the XOR Arbiter PUF variants) as the baseline for comparison. Table 3.5 shows the comparison between our method and Wang's in [59]. Accordingly, to obtain estimated model with 90% prediction accuracy, our method shows to require a significantly smaller number of CRPs for training overall, compared to that of Wang's. We assume that a good proportion of CRP reduction is due to the structure of Mursi's MLP [6] as we reproduced. It is already proven in the experiments in [6] that this MLP requires much less CRPs to converge to an accurate model compared to previous MLP models. This therefore gives advantage to our method where we apply Transfer Learning to Mursi's MLP in order to further decrease the required training data.



(**a**) 2% noise (**b**) 5% noise (**c**) 10% noise

Figure 3.10: Modeling 64-bit 4-XOR Arbiter PUF with and without Transfer Learning with presence of noisy CRP.



(**a**) 2% noise (**b**) 5% noise (**c**) 10% noise

Figure 3.11: Modeling 128-bit 4-XOR Arbiter PUF with and without Transfer Learning with presence of noisy CRP.

Table 3.5: Comparing Wang's Transfer Learning with CNN [59] (A) and our proposed transfer learing plan with MLP (B).

Challenge size	No. stage	Prediction accuracy	No. CRPs $\times 10^3$	
			A	B
64	2	90%	3.5	.8
	3	90%	31	3
	4	70%	32	-
		80%	39	9
		90%	NA	12
128	2	90%	5.5	2
	3	90%	115	8
	4	70%	170	30
		80%	190	35
		90%	380	45

We proposed Transfer Learning technique to model strong PUF with Multi-layer Perceptron (MLP). We proved that a trained MLP model of PUF circuit can be a source domain, and the weight and bias values of the hidden layers of the model can be reused to model a PUF circuit with similar structure with less training data. We experimentally proved that we can considerably decrease the required number of CRPs by approximately 50% compared to an MLP model which is initialized with random values. We also showed that our technique has a small failure-rate close to zero for noise-free modeling and maximum 15% for training with noisy CRPs where 10% of the CRPs are incorrect. We showed that using Transfer Learning on MLPs has resilience to various levels of noise up to 10% of noisy CRPs in the training data.

Next we elaborate on Sub-space modeling as another optimization technique in modeling PUF. Sub-space modeling in turn is a technique mostly focused on the internal structure of the target PUF. In a way, sub-space modeling as discussed before, suggests a divide-and-conquer approach in modeling PUF with dividable structure. In the following section, we explain how this technique can be applied on XOR Arbiter PUFs and we show that with sub-space modeling, we can drastically decrease the required number of CRPs to model the PUF. This in turn can be regarded as an interesting solution solely to be used for modeling strong PUFs. However, it will have it's downsides as well which we will point out to later in the coming chapter. Then in the last chapter, we will talk about how sub-space modeling and transfer learning can potentially co-operate to amend the complexity in modeling strong PUF with very large

structural complexity.

3.4 Subspace Modeling: A Technique to Tackle Requirement for Large CRP data for training

The common approach in building the equivalent software model of PUF is to use Machine Learning (ML) modeling techniques. The idea is to train a probabilistic model which can estimate the CRP characteristic of the PUF with high probability. In this approach, a set of CRPs is captured, and a training algorithm is used to converge the model's characteristic to an estimation of the target PUF's CRP characteristic, according to the captured CRP set [5], [7]. An important challenge in ML-based modeling of PUF is to deal with the structural complexity of the PUF (for instance k-XOR Arbiter PUF with k larger than 4). Usually, the strong PUF with high complexity require significantly large number of CRPs for training [26], [54], [55], [62]–[64]. This on one hand is appealing for the designers' community to implement strong PUF with high complexity to protect against model building attacks. On the other hand, it imposes additional cost for protocols which rely on enrolling the PUF with ML-based modeling. This is due to the fact that CRP collection is done usually during the manufacturing test phase, since doing it post fabrication is an expensive operation. Therefore, spending more time during the test phase to collect large number of CRPs, increases the time of testing and consequently the manufacturing cost. In addition, large CRP training set size leads to spending hours of training time per PUF model. This in turn leads to excessive computation power usage and thus increasing the cost of modeling. These factors generally imply that modeling strong PUF with high complexity using the conventional ML methods is an expensive solution for enrollment. This work is inspired to put sub-space modeling into practice as a cost-efficient solution for enrollment. In sub-space modeling, the assumption is that the designer can access the internal values of strong PUF with large complexity during the test phase. In this way, the designer has multiple modeling targets with reduced complexity, which in turn need fewer CRPs for training compared to the whole PUF. In this work, we show how sub-space modeling can be performed on XOR Arbiter PUF to provide a model with a significantly reduced cost. Our contributions will be to develop an ML-based enrollment solution with the following features:

- Able to generate (at server level) all the possible CRPs of the target PUF.
- Requiring a small amount of memory at server level to store such an information.
- Providing a constant and short enrollment time per PUF, thus applicable in a real industrial/commercial environment.

In turn, sub-space modeling can be a suitable enrollment solution for the designers' community compared to other conventional modeling methods. Given that with ML-based enrollment, access to the full CRP space is provided which can emerge into new protocols for authentication and encryption key generation. Such protocols require also novel approaches to restrict CRP access after enrollment. This is crucial, as it prevents openly accessible CRPs to all parties after enrollment to avoid model-building attacks. Moreover, it is important that the physical access to the I/O PUF is disabled once the PUF is enrolled successfully. An example of that can be found in [48], where it is suggested to permanently disable the physical access-points to the PUF, e.g., by burning irreversible fuses so that other parties cannot access the PUF.

3.4.1 Sub-Space Modeling Method

A schematic of our proposed method is shown in Figure 3.12. Here the illustration shows sub-space modeling for a variant of XOR Arbiter PUF. As shown in Figure 3.12, the internal data from each sub-component (r1 to rk), in conjunction to their corresponding challenge values (e.g. c1 ... ck + r1 for APUF_1, c1 ... ck + r2 for APUF_2, etc.) are fed separately to trainer functions to discretely generate estimation models of each sub-component. After the trainer functions provide accurate estimated models of each sub-component, we merge the sub-models into forming a whole model which represents the whole PUF. We also take the assumptions below on how we can provide data for training:

- The strong PUF structure should be dividable into smaller sub-components with reduced complexity. We assume that these sub-components are themselves functions of the input challenge to the PUF.
- An accompanying hardware extractor should be provided which has physical access to the internal sub-components' I/O. The extractor can capture the value of the internal sub-components in addition to the response of the whole-PUF, for any given challenge.

Table 3.6: LR training hyper-parameters

Inverse regularization (C)	**Tolerance for stopping (tol)**	**Max iter**	**Solver**
1.0	0.0000001	10000	liblinear

- We represent CRvP for samples taken from PUF and its internal data. A CRvP comprises a Challenge vector, and Response vector (RvP) which is a vector of values comprising the responses of internal sub-components as well as the response of the whole PUF.

- Number of sub-components addressed by the extractor are enumerable. We assume that the connectivity of the internal values to the extractor does not disturb the design and functionality of the PUF itself.

- Once the enrollment is successful and the accurate model is stored, the physical access to the internal values within the PUF circuit is permanently removed. This is essential to prevent future threads which may be able to regain access to the internal values via the extractor.

We also measure the accuracy of the model before storing it on sever. We compare the model's predicted responses with the PUF responses for a set of challenges in a CRP set dedicated for testing (different from the ones used for training). After testing the model's prediction accuracy, it is stored for the given strong PUF on the verifier server for future use.

3.4.2 Experimental Setup

Again, here we used a python-based simulator of XOR Arbiter PUF [51]. We used the simulator to generate 10 instances of 128-stage 2, 3, 4,5,6,7,8,9 and 10-XOR Arbiter PUF variants. For each variant we generated 100,000 CRvPs, therefore a total of 9 million CRvPs for all the instances of all the variants have been generated. Noting that in this simulation we did not model the PUF instability noise explicitly. Therefore, the randomness is only due to the signal propagation delay as we discussed earlier.

For modeling we used Sklearn's Logistic Regression (LR) to model each sub-component independently. The reason we chose LR is that comparing to other modeling techniques, LR shows to converge considerably faster, using less computation power. Also,

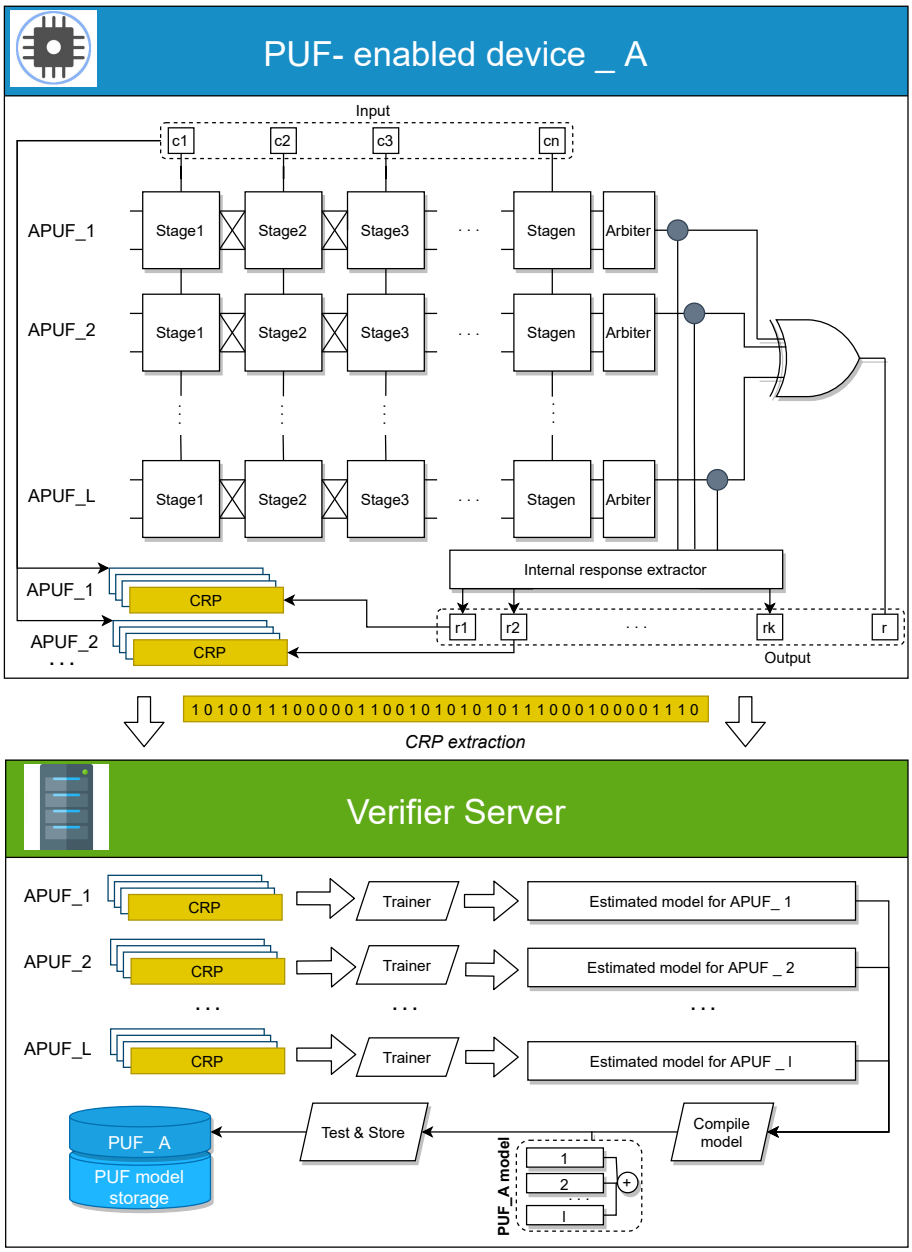


Figure 3.12: Showing how sub-space modeling can be used for strong PUF modeling with separable components. Here the PUF variant is a n-stage k-XOR Arbiter PUF.

	XOR size															
	4				5				6				7			
	css(MB)	ϵ	T(h)	ms(KB)	css(MB)	ϵ	T(h)	ms(KB)	css(MB)	ϵ	T(h)	ms(KB)	css(MB)	ϵ	T(h)	ms(KB)
R	0.377	99%	2:52	4.1	7.9	99%	16:36	5.1	N/A				NA			
T	3.1	98%	0:02	4.1	34.6	98%	00:12	5.1	236.2	98%	4:45	6.1	629.8	98%	66:53	7.2
M*	15.7	95%	<0:01	10.6	15.7	95%	<0:01	25.3	157.4	95%	<0:01	67.1	472.4	95%	0:02	199.7
S	0.944	98%	0:05	180.2	6.3	97%	1:5	1280	18.9	97%	6:1	3072	44.1	96%	18:2	10752
SS	**0.490**	**98%**	**<0:01**	**4.1**	**1.48**	**97%**	**<0:01**	**5.1**	**1.16**	**97%**	**<0:01**	**6.1**	**1.5**	**96%**	**<0:01**	**7.2**

Table 3.7: A comparison of cost of training in modeling variants of XOR Arbiter PUF. Here SS refers to our proposed sub-space modeling method. R is the modeling method used in [26], T is the modeling method used in [55], M* is the modeling method first proposed in [6] and then revisited in [64]. S also is the modeling method used in [63].

LR seems to be a good starting point to explore modeling techniques due to the fact that LR is relatively the simplest modeling technique compared to others such as Artificial Neural Networks or Support Vector Machine. In terms of the training specifications and hyper parameters, our entire parametric consideration for training with LR are given in Table 3.6. Noting in addition, that we measure css in terms of bytes for n-stage k-XOR Arbiter PUF as in equation (3.2) where N is the number of CRPs in a given training set.

$$css_{(byte)} = \frac{N \times (n + k + 1)}{8} \tag{3.2}$$

3.4.3 Experimental Results

First we measure the cost of training using sub-space modeling, and compare it to some of the known and recent modeling methods. Table**??** shows the cost of training with respect to various k in n-stage k-XOR Arbiter PUF. Here we compare our sub-space modeling method (SS), to other modeling methods practiced for XOR Arbiter PUF modeling. Noting that the competing methods (R,T,M* and S) in Table. **??** are not optimized for enrollment. These methods are generally proposed to model the whole XOR Arbiter PUF using a back-propagation technique to adjust the internal values θ directly according to the response behavior of the whole PUF model. Therefore they do not consider the internal responses.

From Table 3.7, R is the modeling solution of [26] which uses Logistic Regression with RMSProp as the training function. In T [55] also, the modeling approach is the same as R, while the training is optimized to be faster. In M* [6] and S [63], Artificial Neural Networks are used as the underlying model and Adam optimizer as the training function.

As shown in Table 3.7, sub-space modeling indeed is capable of reducing the cost

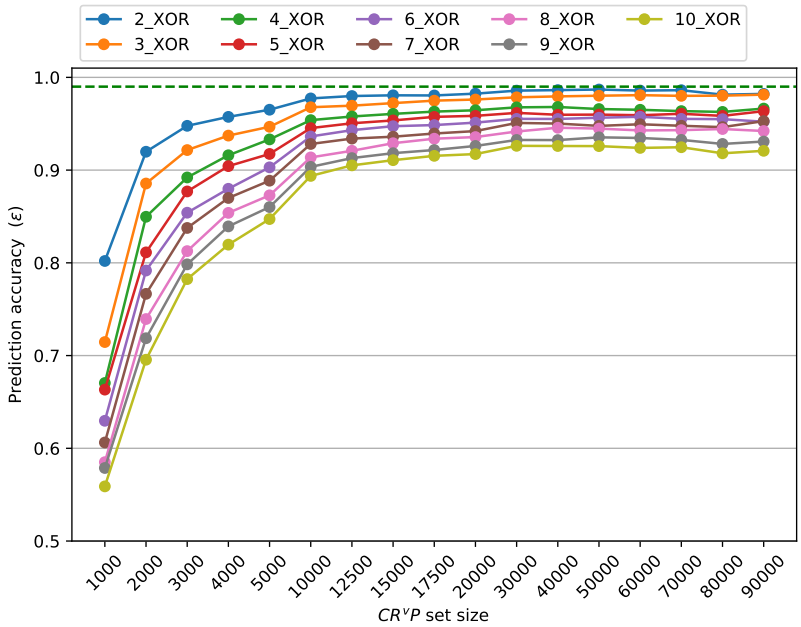


Figure 3.13: Illustration showing the convergence of prediction accuracy ϵ of the XOR PUF variants with respect to increasing css using sub-space modeling.

of training. Our solution (SS) seems to have the highest efficiency in terms of css, especially for modeling 5 6 and 7 XOR sizes. The exception is for 4-XOR where R shows to have a better result in terms of css. Given however that for the same XOR size, our solution takes significantly less training time T compared to R. The closer case in terms of css to our solution, is S. However, in terms of T and model size ms, our solution shows better results overall. In terms of T, our model seems to have overall the best performance as well. Given also that closer case in terms of T to our model is $M*$. However, for $M*$, the css seems to be considerably larger. It can thus be implied here that sub-space modeling overall can be considered to have the highest efficiency in terms of all factors that constitute the cost of modeling, for modeling strong PUF with increased complexity.

Next in sub-space modeling, we measure the prediction accuracy with respect to increasing the number of CRvPs for a larger scale of XOR Arbiter PUF variants. Figure 3.13 shows the evolution of prediction accuracy ϵ with respect to increasing CRvP set size for XOR sizes 2, 3, 4,5,6,7,8,9 and 10. We observe that the convergence points for ϵ degrades proportionally with increasing XOR size k. For instance, for 4-XOR variants the convergence point for ϵ is at 0.98 while for 10-XOR it is at 0.93. Another conclusion

is that ϵ for all variants start to converge to its maximum value at around 10,000 CRvPs. The peak for ϵ could be seen in the range of 10,000 to 40,000 CRvPs. While from 40,000 CRvPs above, the ϵ seems to degrade slightly. Which could be due to overfitting the model with too many CRvPs. This means that sub-space modeling at this stage has a downside which is the degradation in the maximum prediction accuracy with increasing PUF complexity.

To further analyze prediction accuracy degradation, we measured the distribution of ϵ over all the sub-components of all the variants of k-XOR for k in 2,3,4,5,6,7,8,9 and 10. Figure 15 shows the histogram of all the sub-model's prediction accuracy with respect to several CRvP set sizes. We observe that the prediction accuracy is less concentrated for smaller CRvP set sizes like 1000 or 2000. This easily justifies the low prediction accuracy of the whole model. Since the rate of miss-prediction at these CRvP set sizes are quite high, and as we know that the accuracy of the whole model can be defined as a product of the accuracy of the sub-models. Therefore, the effect of internal miss-prediction on the whole model's prediction accuracy will be significant. For instance, for a 2-XOR Arbiter PUF we have:

$$\epsilon_{WPUF} = \overbrace{(\epsilon_{SPUF_1} \times \epsilon_{SPUF_2})}^{\text{Probability of two correctly predicted responses at the same time}} + \underbrace{((1-\epsilon_{SPUF_1}) \times (1-\epsilon_{SPUF_2}))}_{\text{Probability of two incorrectly predicted responses at the same time}} \tag{3.3}$$

where ϵWPUF refers to the prediction accuracy ϵ of the whole PUF model, and ϵSPUF_i refers to the prediction accuracy ϵ of the ith sub-model. Given also that this equation extends for larger XORs, where there needs to be computed the product of miss-prediction probability of every even number of sub-models. Here in equation (3.3), if we give the average prediction accuracy to the parameter ϵSPUF_i, it yields approximately the prediction accuracy value for the whole model as indicated in Figure 14.

Looking at Figure 3.14(e), it is apparent that for the 40,000 CRvPs where the peak value of ϵ is achievable, the ϵ values are distributed around 0.992. This also justifies why the accuracy for modeling variants of k-XOR with increasing k, degrades as well. Again, according to equation (3.3), variation as small as 0.01% in the prediction accuracy of the sub-space models, can lead to variation of up to 1% of the whole model. For instance, according to an extended version of equation (3.3) for 10-XOR, we need in average,

99.9% prediction accuracy achieved for each sub-model to then achieve 99% prediction accuracy for the whole model. However, with our observation of the prediction accuracy of models trained with the most optimal training set size, the sub-space models' prediction accuracy is not concentrated on 99.9%. For larger than 40,000 CRvPs, it appears that we overfitted a population of sub-models (see the outlined area in Figure 3.14(f)) which deteriorated the estimation of the whole model's prediction accuracy. Figure 3.14(f) shows that a considerable number of models trained with 90,000 CRvPs, appear to have prediction accuracy lesser than the median 99.2% at 40,000 CRvPs. This means that only increasing the CRvP set size is not a solution to achieve the highest accuracy for the whole model, as it can lead to overfitting the sub-models.

Our general observation on the sub-space modeling here infers that sub-space modeling seems to be a considerably resource efficient modeling technique. Given however, that its overall accuracy is more sensitive to minor prediction accuracy variations in the sub-models. Therefore, special care should be given to stabilize the prediction accuracy of the sub-model on the maximum achievable accuracy. For instance, if the maximum achievable accuracy is 99.9%, we set this as the target accuracy for all the sub-space models. This then requires that each sub-space model is trained with discrete attention to the number of CRvPs in order to obtain exactly the target prediction accuracy for the whole PUF model.

Moreover, the number of the sub-components seem to affect the prediction accuracy of the whole model. It is observable that with larger number of sub-components, the accumulation of the probability of miss-prediction errors of the sub-models lead to higher values. Therefore, it is important to manage the number of sub-models. For future extensions of the work, it is potential to try with various dividing factors for sub-space modeling. One example is to divide a k-XOR Arbiter PUF with k being an even number, to k/2 of 2-XOR PUFs and model each 2-XOR Arbiter PUF separately.

Here we presented a technique for modeling strong PUF, using captured internal data of the sub-components of the PUF with high complexity. We showed that sub-space modeling requires significantly less data in order to yield an accurate estimated model of the PUF. For instance, we showed that it is possible to model 128-stage 10-XOR Arbiter PUF with 93% prediction accuracy using 40,000 CRvPs which is equivalent to 683 KB of data. This proved that sub-space modeling is a potential cost-efficient solution for the designers' community whose priority for enrolling strong PUF is to provide an estimation model of the PUF. However, we observed that the maximum achievable

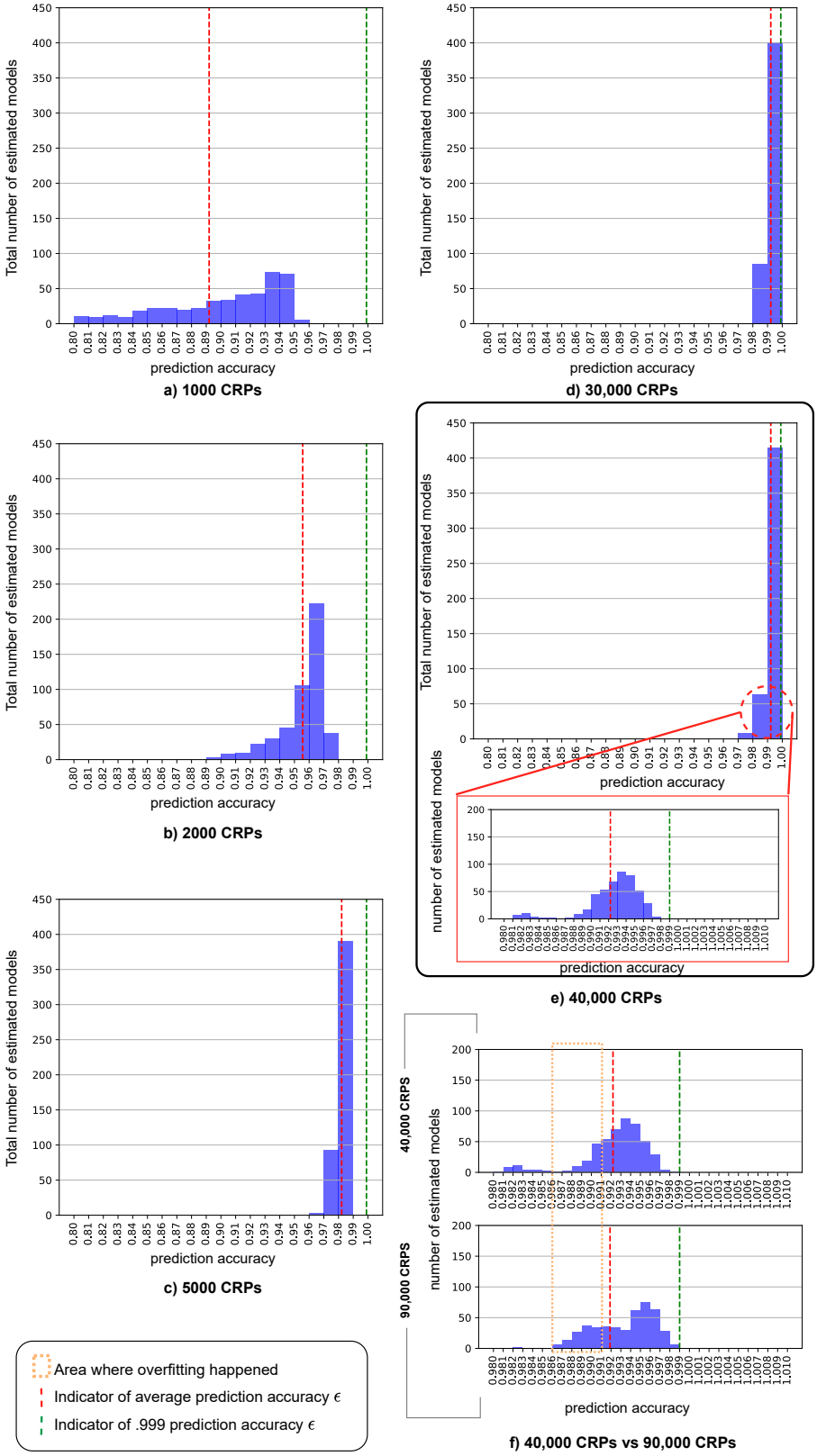


Figure 3.14: Histograms showing the distribution of prediction accuracy ϵ for various training CRP set sizes.

accuracy can be limited with sub-space modeling, due to the internal models' prediction accuracy variation. As expected, we observed that even a small prediction accuracy variation of around 0.1% in the internal models can cause the accuracy degradation of the whole model up to 1%. Depending on the complexity of the whole PUF also, this degradation can be magnified. And moreover, only adding more training data showed not to be the solution to overcome the prediction accuracy degradation. Therefore, this would be an open problem in sub-space modeling of strong PUF with high complexity.

3.5 Conclusion

In this chapter we discussed the modeling of PUF for enrollment. We elaborated on the details and parameters of the training loop for modeling PUF, the training hyper-parameters, and the control parameters. We discussed that it is important to extract the optimal control and hyper-parameter values before beginning to enroll a large group of PUFs. As a primary essential factor, we need to know the number of CRPs we need to train the PUF samples. Depending on the priority, we can target capturing a large set of CRPs, or a compact set of CRPs. We showed that increasing the number of CRPs in turn assures that majority-to-all the PUF samples can be modelled successfully. While on the other side, it enables the necessity to spend more time during CRP-readout to capture a large set. Then, we showed that the primary evaluation phase can unveil to us the minimum CRP set size that can yield in modeling majority of the PUF samples, with a minor chance of failure for a selective number of PUF samples. We discussed that this alternative choice would allow capturing considerably smaller set of CRPs during readout, and lending the time of processing to the training phase where we expect more trial and errors and training epochs may be needed to yield PUF models with desired accuracy. We also discussed that since the training phase can be considered an offline process, and not necessarily a process running during the test phase, we can assume the tolerance for taking more time on training and enrollment in general is considerably higher.

Later in the chapter we discussed two optimization methods for PUF modeling. We discussed Transfer Learning, a trending technique in machine learning to reduce training cost, and sub-space modeling, a structural approach to reduce the complexity of the PUF subjects for training and following that, to facilitate the training with less opposing cost in terms of time and training data.

We explained that in the eco-system of PUF enrollment using ML, Transfer Learning has a lot of potential due to the fact that PUF samples in the same device family share features such as their structure, the distribution of the delay parameters, etc. Since transfer learning benefits from similarity of two training subjects, it can be used here with the goal to reduce the training cost in terms of data and training time. We showed that we can re-use the lateral features of a trained MLP model of a PUF sample, by transferring the weight values into a new MLP model with the same structural spec, targeting to model a new PUF sample with similar structure. Our results showed that TL indeed reduces the training dataset size compared to each time using a randomly initialized MLP model for training.

Then we explained sub-space modeling and emphasized on the potential of the method to model strong PUFs with increased structural complexity, such as XOR PUF with large number of XORs. To perform Sub-space modeling, we clarified that we need access to the internal responses of the PUF. This way we could extract the smaller compartments of the large PUF structure (such as an arbiter chain of an XOR Arbiter PUF) and model it separately. Doing so however, we needed to emphasize as well on the necessity that the access to the internal responses should be eliminated to avoid giving leverage to physical attacks in modeling the PUF with the same facilitated way. Our primary results of sub-space modeling also were promising. We could expectedly lower the training dataset size by a considerable amount.

The methodology and the optimization techniques discussed in this chapter should in turn give us a fast and robust PUF enrollment methodology based on ML. Nonetheless, for each of the optimization methods we discussed, we also observed some short-comings which needs addressing. For instance, use the TL method, we observed for some cases the TL could not lower the training dataset size compared to the case of randomly initializing the model. For sub-space modeling also, we observed that as we increase the granularity of the modeling subject, the achievable accuracy cap is lowered. In the last chapter of this work, we elaborate on these short-comings and propose briefly what we can do to resolve them.

4

Implementation of PUF-based Protocols

This chapter is dedicated to Protocol design. Two of our published papers [65], [66] will be presented in this chapter. At the early stage of the chapter, we elaborate on Fuzzy Extractor based key generation protocol and the vulnerability of one of its derivatives, the Robust Fuzzy Extractor-like method against Helper Data Manipulation attack. Then we explain our proposed intrinsic masking countermeasure to mitigate the vulnerability. This contribution was our first attempt in stepping into protocol design. In that, we do not venture with machine learning. Rather we look at the security of an existing protocol, and how we can enhance it using existing materials. Then in the second phase of the chapter, we elaborate on our novel authentication and key exchange protocol, within which we use machine learning. There set focus not only on the security of the protocol, but also on the possibility of integrating a predictive model of PUF on the server side, and how we aim to handle it's mis-prediction while at the same time taking advantage of the full CRP access it provides.We also briefly explain the possibility we ventured in lightly to emulate strong PUF over a weak PUF substrate. This attempt was meant to allow us to then provide applicability of novel protocol for weak PUFs as well.

4.1 Introduction

As for all the security protocols, keeping the security and the reliability of the secret values that are used for key generation or identifiers at maximum is the priority. To do so, the primary protocols suggested using parity checks, which is one of the pioneering techniques to assess the reliability of a binary code. However more complicated coding schemes emerged over the past decades and new mechanism such as hashing also were

added to the circle of coding techniques which ensure not only the underlying secret values, but also their confidentiality, ensuring that the value has been inspected only by authorized individuals.

Ensuring that a secret value is secure and reliable is one challenge to overcome, to recover it from noisy or unwanted manipulation is another challenge. This is a normal case in situations where it is expected that remote parties who are communicating with each other and demand a confidential exchange of data, need to setup an encrypted channel wherein the exchanged confidential data is encrypted by a mutual key. Of course, there are symmetric and asymmetric encryption techniques, the mutual criterion in all the variants is to ensure that at all times the same secret value which has been used for encryption, is available for decrypting the data for later use.

We know already that PUF is a noisy source. At its best characteristic, we still expect that some CRPs have a flipped response. To ensure a reliable secret value that is originally generated and then re-generated from PUF is used, a method called Fuzzy Extraction (FE) was introduced. Briefly explaining, FE-based methods enable recovering an original value from a noisy source based on a code-offset which is a product of a recoverable codeword and the original value. The code-offset is a publicly available piece of information that can be queried by any party acclaiming to be authentic. This is the initial and still a conventional recovery method that is used for PUF-based encryption algorithms. But it has its flaws, including susceptibility to helper data manipulation attacks (HDM attacks). The way these attacks work, as we mentioned briefly earlier, is to modify the code-offset in a way that the decoding function is forced to decode and recover codewords that are easily guessable by the attacker. Since we set goal to venture into protocol design based on PUF, we decided first to tinker how we can secure FE-based techniques(Especially the RFE-like methods which are a lighter version of FE-based methods) against HDM attacks. Following that, here we discuss on a masking technique we developed to protect the RFE-like techniques against HDM attacks. However, in this countermeasure, we did not salvage any of the potential of ML that we discussed earlier to come to use in PUF-based encryption. It is mostly due to the fact that the underlying PUF type in this trend of work we ventured in is a weak PUF.

Therefore, we still have the potential to realize a reliable and secure key-exchange and/or authentication protocol that can use the potentiality of ML-based model of PUF. To realize that we discuss here as well on a novel key exchange and authentication protocol we designed which utilizes the ML-based model of PUF and creates a repetition-

like coding technique that is locally recoverable. As we explain later in this chapter, this protocol is designed with the aim to completely remove the necessity of a helper data, a point we learned from the earlier study we had on the RFE-like encryption methods and their vulnerability against the HDM attacks.

4.2 Introducing Helper Data Masking: An Improvement for FE-based Key Exchange Protocols

The primary part of encryption key generation is to read the power-up binary values of memory cells. During the enrollment phase, the power-up binary values of memory cells are captured and stored on the server. These values are in turn the source to create encryption key values. An original key value is the base value which is hashed to create the encryption key on the server side. Meanwhile, an offset code from the original key value is also created as the helper data to send to PUF-enabled device. The PUF-equipped device uses the helper data in a fuzzy extractor to mutually generate the original key value known by the server as well. In this section, we will explain the *Robust Fuzzy Extractor*-like (RFE-like) construction as discussed in [67]. RFE is commonly used in key generation schemes to re-generate an original secret value from a noisy source (e.g., PUF) by using a publicly available helper data. To ensure a secure value recovery in an RFE construction, a 2 step verification is performed. First, a hash value generated from the recovered secret value is compared with a hash value given as part of the helper data. These two values should be the same to succeed in the first step. Secondly, the Hamming Distance (HD) between the recovered value and the regenerated value is checked with a pre-defined threshold. The HD value lesser than the threshold implies success in the second step. If any of the verification steps fail, the key generation is considered a failure.

RFE-like construction is also a derivative of RFE construction with the difference that the security of the RFE-like is provided with only the verification procedure through a comparison of an original and a regenerated hash value. A detailed description of an RFE-like construction is in the following.

Figure 4.1.(a) shows the enrollment phase of an RFE-like construction where a verification hash value h is created by hashing an SRAM-PUF response w with $H1$ hash function. Helper data s is also created by XORing the SRAM-PUF response w with a

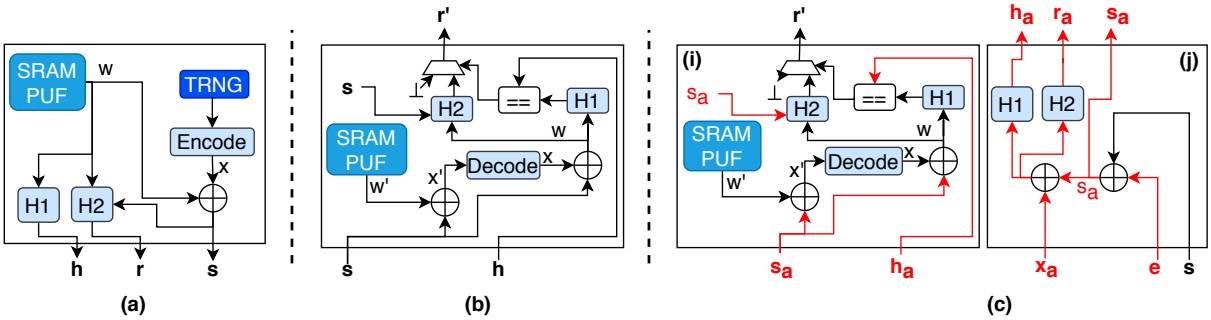


Figure 4.1: Illustration showing the RFE-like construction (a) the registration phase, (b) the recovery phase, and (c) the recovery phase with presence of HDMA, of the SRAM-PUF based encryption key generation.

generated random codeword x. Noting that a *True Random Number Generator* (*TRNG*) is used here to generate a random message which is encoded to yield a random codeword x. The SRAM-PUF response is hashed with $H2$ to extract the encryption key r which is stored on the server to be used during the encryption key recovery phase.

Figure 4.1.(b) shows the recovery phase of an RFE-like construction where helper data s and verification hash value h are requested from the server by the PUF-enabled device. As the device receives the queried data s and h, s is then used to recover the generated codeword x from the noisy SRAM-PUF response w', and then the original SRAM-PUF response w is recovered by XORing the x with s. The recovered w is then hashed with $H1$ and the regenerated hash value is compared with h that was received from the server, to verify the originality of w. If equal, w and helper data s are hashed with $H2$ to regenerate the encryption key r for final output r'. If not original, the extractor returns failed on r'.

4.2.1 Helper Data Manipulation Attack

HDM Attack refers to an attack against soft decision ECCs [68], wherein the attempt is to reconstruct the original PUF response in a divide and conquer fashion, by passing many attempts of sending modified helper data to the PUF-enabled device. During each failure in response reconstruction, the adversary learns information about the original response, leading to the reconstruction of the response. While proved in theory, in practice, it will take many attempts until the adversary obtains full knowledge of the original response. In contrast to the primary HDMA, another HDMA was also introduced in [67] where the attempt is not the reconstruction of the original PUF re-

sponse but instead to try to set the reconstructed PUF response to a value known by the adversary.

In practice, often single long codes are not used as codewords, but instead, it actually consists of smaller concatenated codewords [69]–[72]. This approach is considered for the RFE-like construction we use in our evaluations. In this case, the first assumption is that the adversary knows the size of target codeword, and will apply helper data modifications repetitively on each sub-division of the helper data. Figure 4.1.c shows the schematic of HDMA during the recovery phase. During the attack, we assume that the adversary is eavesdropping on the public data query and responds to the query from the server. Using helper data s and hash value h from the server, the adversary targets a subsection of the helper data s, by applying an error vector to each sub-part to manipulate the helper data and create *adversary's helper data*, s_a. The adversary also has a reduced set of codewords C_a which is a subset of the larger codeword set C that is used on the server during the enrollment phase to create helper data. This reduced set C_a is in fact extracted relative to the error vector the adversary is using. By choosing a codeword X_a from C_a, the adversary attempts the re-generating of a PUF response w_a, which in turn is used to create a candidate adversary verification hash value h_a and adversary encryption key r_a.

The adversary's' helper data s_a and adversary's' verification hash value h_a are then sent to PUF-enabled device, as a response to the public data query the PUF-enabled device has passed initially. While on the PUF-enabled device, the same steps of the recovery phase are taken, the way for the adversary to know if the re-generation of adversary's key value r_a is successful, is to receive the positive response of the comparison check of the adversary verification hash h_a with the one locally generated on PUF-enabled device.

Usually with the presence of HDMA, the failure rate of key generation will rise as expected. Therefore, the recovery process will be to repeat, until either the number of attempts for re-generation per query passes a pre-defined threshold, or the regeneration of a mutual key between the adversary and the PUF-device becomes successful, leading to the result of having r' being equal to r_a.

As the RFE-like construction is explained, one can see that the reason why HDMA can be successful is that there is no check on the authenticity of the received helper data s during the recovery phase. In other words, during the recovery phase, any helper data $\bar{s}$ can be used to recover a codeword $\bar{x}$ and following that, produce a random PUF image

$\bar{w}$ and a verification hash value $\bar{h}$. And as long as $\bar{h}$ is equal to a received verification hash value h, the regenerated key r' is valid. And since the adversary can also decide on the value of h, producing h_a, he can enforce a match between a $\bar{h}$ and h_a after multiple trial and errors.

4.2.2 Helper Data Masking ith Variable Positioning

In this section, we elaborate on PUF-based helper data masking mechanism with variable positioning. First, we discuss how we employ PUF as the source of mask value generation. Then, we elaborate on the variable positioning mechanism. Later on, we discuss the threat model which is a derivative of HDM Attack that has knowledge of masked helper data with variable positioning. We then statistically show how the new threat model still has a large guessing field to explore breaking the key, compared to the conventional case of attacking helper data without masking as discussed in [67].

4.2.3 Mask Vector Generation

Commonly on a secured PUF-enabled device, using a non-volatile memory (NVM) on the PUF-enabled device to store secret values (e.g., mask values) is not suggested. That is due to its cost and the security issues with this type of memory. Moreover, for generating mask values on a PUF-enabled device, the values should be highly randomized to avoid any exploitable leaks that allow adversarial third parties to recreate the mask.

Here we propose using SRAM-PUF itself as a source of mask generation. In our proposed scheme, the SRAM-PUF will be the source of both the key generation, and mask vector generation. This in turn eliminates the need for any storage component on the board to store the mask values. Therefore, the method will be cost-efficient and physically secure. Moreover, assuming the PUF has a good characteristic, is an ideal source of randomized value generation. That is a key criterion for key generation which makes at first place the PUF a good candidate for encryption key generation. For the same reason, it can be a good candidate for generating mask values with high randomness as well. Here we define the mask vector generation as a process to read the power-up binary values of several consecutive memory cells from an SRAM device.

Using PUF as the mask value source, one should guarantee that the generated mask vector is highly reliable since the source is inherently noisy. This means that the recovery of the mask value on the PUF side should yield exactly the same value used on the

server side. To assure the reliability of mask vectors, one can consider using a fuzzy extractor that can suitably address the noisiness of the PUF source. The helper data based RFE-like construction in this case can also be used to assure reliability, equally to that in Key generation.

To elaborate on helper data masking, let us define the process of masking to be the XORing of a candidate mask vector on some sub-parts of the helper data bit stream. We assume the mask vector itself is a bit stream as well. The masking process is issued on the server side after the helper data is reloaded and just before answering the query from the PUF-enabled device. To answer the query, the server sends the masked helper data with an extended helper data for the recreation of the mask value alongside with the rest of the public data to the PUF-enabled device for mutual key generation. On the PUF enabled device, the masked helper data is demasked using a mask vector that is recovered from the on-board PUF and the extended helper data.

For an adversary in the middle, we assume that there is no access to the source of the mask value. Thus, the adversary has to undergo a guess-based procedure to discover the mask vector value in order to demask the helper data. Recalling that the adversary initially needs the original helper data in order to recreate the PUF response. Therefore, one can say that masking of helper data potentially increases complexity of the HDM Attack. This in turn can distance the adversary from the point of success in generating a mutual key.

The general sketch of our proposed key generation protected with helper data masking is shown in Figure 4.2. The additional part is the masking of the helper data before answering a query from a device. Here also the helper data based RFE-like construction is used for mask vector generation during the recovery phase on the PUF-Enabled device. Following Figure 16, during the registration phase, aside to enrolling PUF response w_1 for key generation, the PUF response w_2 as mask vector generation is also enrolled. Correspondingly, helper data s_2 of the enrolled mask vector is also created and also w_2 as the original mask vector, is XORed with s_1 to mask the primary helper data and produce Ms as masked primary helper data. All of the generated information in this phase are then stored on the server.

During the recovery phase, the public information sent to PUF-Enabled device includes 2-parts helper data, where the initiative part is the masked primary helper data Ms used for key generation. The latter part is the secondary helper data $s2$ used for mask vector recovery. During the recovery phase, the primary attempt is to recover the origi-

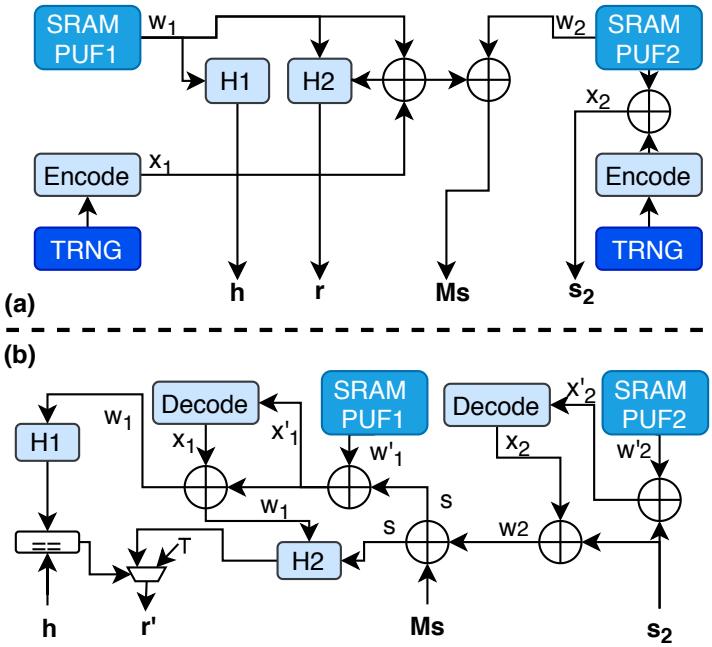


Figure 4.2: Illustration showing the structure of helper data based RFE-like construction protected with helper data masking. Here (a) is the registration phase, and (b) is the recovery phase.

nal mask vector. Thus, secondary helper data $s2$ is XORed with captured PUF response $w'2$ from SRAM-PUF 2, and the output is decoded to generate a recovered codeword $x2$. The output is then XORed with $s2$ to recover $w2$ as the original mask vector. $w2$ is then XORed with the masked primary helper data Ms to demask the helper data and produce s.

Using this construction, one can assure that reliability is equally provided for the recovered mask vector as well as in PUF response for key generation, in addition to primarily securing primary helper data. However, the secondary helper data is now exposed and HDM Attack can exploit that to break the masking. However, with the extension of adding variable positioning mechanism to the masking scheme, one can assure that even with the exposure of the mask value helper data, the adversary has to undergo an exploration in a large guess-field in order to guess the position of the mask and correctly demask the helper data.

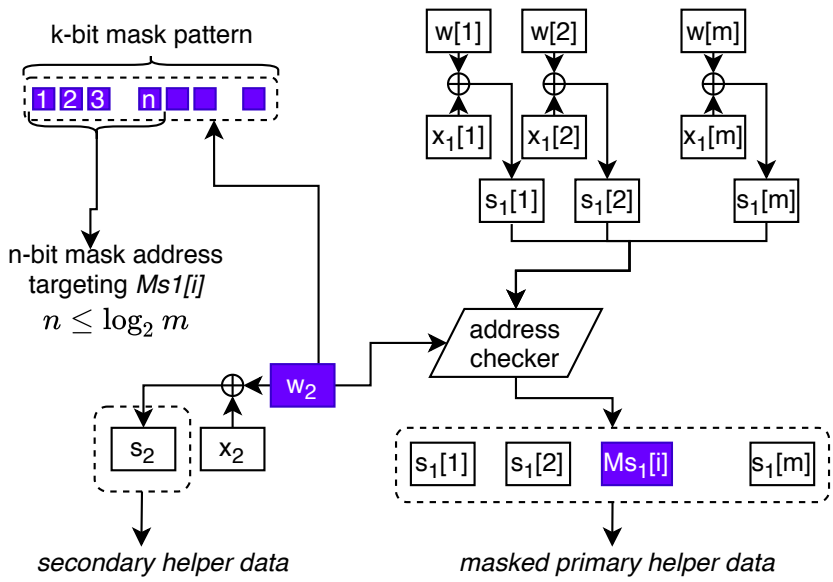


Figure 4.3: Illustration showing the schematic of our proposed code-offset masking.

4.2.4 Masking with Variable Positioning

To mask the helper data with variable positioning, we propose applying a mask vector which is a bit vector smaller than the helper data bit string. Additionally, we propose to variate the mask vector's application point with respect to the value of the mask. Therefore, the address of the target region on the helper data string is defined by an address vector that is defined within the mask vector itself. Figure 4.3 shows how this positioning mechanism works. Wherein the first n bits of a k-bit mask vector, $w2$ is used to define the address of the mask vector (e.g., the ith block of the primary helper data $s1$ as shown in Figure 4.2). Given that the helper data and the mask vector are bit strings, the address checker block shown in the figure can in turn be a shift register which outputs the bit string with the same size of the helper data. In the string, the mask vector is shifted n times from the beginning of the string to be placed in the required address for masking. The output string then can be XORed with the helper data to produce masked helper data.

Following this construction, the address of the target block depends on the true value of the mask vector. In specific, it is depended on the part of the mask vector which the address is extracted from. Moreover, to break the key from a key generation procedure protected with masking with variable positioning, the HDM Attack model needs to adapt as well. In the following we explain the HDM Attack that has knowledge of masking with variable positioning.

4.2.5 Vulnerability of Masking against HDM Attack

With the application of masking over helper data, the HDM Attack as proposed by [67] will not be applicable anymore to break the key. However realistically, we can assume that at some point the adversary will obtain knowledge that the helper data is masked. We assume here the worst case in which the mechanism of helper data masking with variable positioning is known to the adversary. Thus, the adversary will try at the same time to find a combination of guessable values for the codeword with reduced entropy for mask value regeneration, the position where the mask is applied on the masked helper data, and the codewords with reduced entropy for the regeneration of the encryption key. The key point here is that with reducing the entropy of guessable codeword to break the mask value, the adversary is still faced with the same entropy of guessing the position of the masked region on the helper data. In other words, since the position of the masked region is dependent on the true value of the mask, its entropy will stay the same even after modifying the helper data by the adversary. In the following, we explain in details the threat model against masked helper data. The new threat model of HDM Attack which is a derivative of Becker's HDM Attack proposed in [67] is shown in Figure 4.4. The primary phase of the new HDM Attack is to modify the mask value. At this phase the adversary first elects an error vector ei and accordingly a Xa, i form a reduced set of codewords with k possible codewords, accordingly $Xa, 1, ..., Xa, k$. The helper data s_2 for the mask value is then XORed with the elected ei and Xaj. At this stage the adversary's modified mask value $wa2$ is generated. $wa2$ is then passed into a mask stream generator function which takes in addition an elected $Addri$ which is a number indicating the position of the mask on the stream. The mask stream is then XORed with the received masked helper data Ms. The product of the last XOR is an elected demasked helper data $dmsa$ which goes into phase two of modification.

As can be seen, in this model, two guessing fields in phase 1 are similar to that of phase 2. Specifically, the guessing field for electing an error vector and a codeword. However, an additional guessing field is also needed to elect the address of the mask. Noting that the guessing field for the address of the mask cannot be reduced similar to the guess field for the reduced codewords to elect a Xa_i. This is due to the fact that the address of the masked region depends on the original value of the mask. Recalling that we proposed to define the address space within the mask value itself (e.g., the first n bits of the mask value). Therefore, at any case, the adversary needs to brute-force

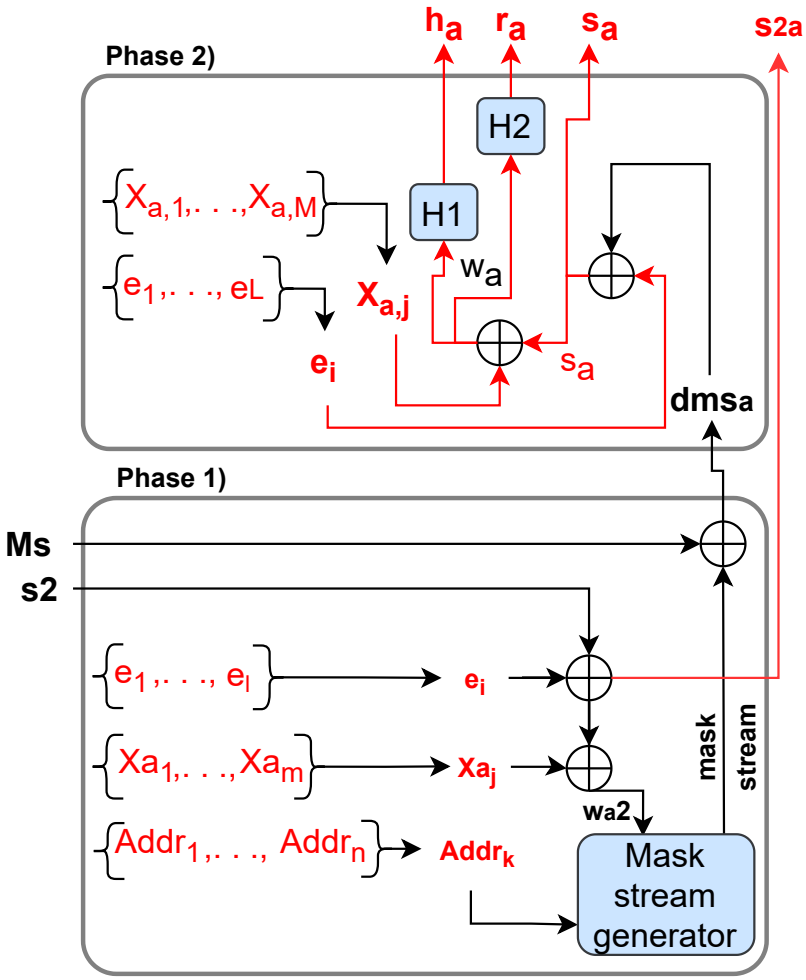


Figure 4.4: Illustration showing the structure of HDM Attack with a new guess field for the address of the masked region on the helper data. Mentioned Addrk is an elected value from the possible positions of the masked region on the masked helper data Ms.

the guessing of the address value. This means that if a user initially defines a large address space for masking the helper data, the guessing field for the address value on the adversary's side would consequently be large proportionally.

We recall that the intention of masking the helper data is for further randomization of the helper data stream. However, at some point, the mask value could result in a neutralized product which could make the masking ineffective. Such a case can appear if the product of the mask vector on the specified region of the helper data would yield a product where the affected codeword(s) of that region are new codewords. This in turn means that the baseline HDM model can break the key without going through phase 1 as shown in Figure 4.3. In the following, we elaborate on a process to detect

such mask values as we refer to as weak mask vectors, in order to build a more robust masking scheme.

4.2.6 Detection of Weak Mask Vector

In a noise-free setting for the SRAM PUF source for mask vector generation, the first milestone is the selection of mask vectors. In the specific case against HDMA, not any arbitrary value for a mask vector is secure. Potentially against HDMA, certain values of mask vectors would still allow HDMA the equal chance of success as in the case of no masking, if they fit in the equation brought in (4.1). We refer to these mask vectors as weak mask vectors.

$$\begin{aligned} &C_a \subset C \\ &\forall c_i \in C_a, \exists m \Rightarrow m \oplus c_i = c_i' \; ; \; c_i' \in C_a \end{aligned} \tag{4.1}$$

wherein c_i and c_i' are codeword elements of the main codeword set C and C_a is a subset of C that is used by an adversary.

Against such mask vectors, the adversary can yield success without attempting to demask the helper data. Accordingly, he first queries the server, receiving a masked helper data Ms and a verification hash value h. Then, XORs error vector e with the masked helper data to create Msa. Noting that the adversary has no knowledge of helper data being masked. In XORing a candidate codeword Xai with Msa, the adversary then creates a predictable PUF response wa which then is passed to Hash function $H1$ to produce adversary's verification hash value ha.

On the PUF-enabled device, the pair of Msa and ha are given. Primarily, the helper data is demasked by XORing the locally regenerated mask vector m, and adversary's helper data sa is resulted. Then, the helper data is XORed with captured PUF response w' and noisy code x' is produced which is then passed to the decoder function to produce recovered codeword x. At this point, the mask vector will be called weak, if the recovered codeword x, assuming to be one of the guessable codewords by the adversary, is itself a combination of the mask vector and a codeword guessable by the adversary. In this case, after the final XORing of helper data sa and the codeword to generate recovered PUF response, it will result in the same wa computed on adversary's side, which will lead to producing the same verification ha as sent by the adversary to the

PUF-enabled device. Therefore, at the end, a mutual key will be generated between the adversary and PUF-enabled device, despite using helper data masking to prevent it. There in helper data masking, it would be necessary to avoid considering a potential weak mask vector.

Going back to (4.1) to identify the weak mask vectors, the equation in fact simply suggests that if a candidate mask vector in XORing with a predicable codeword from C_a lead to another predictable codeword from the same set, then the mask application on Helper Data is not effective against first order HDM Attack. In this case, we address the Becker's baseline HDM Attack as the first order HDM Attack.

However, in order to utilize equation (4.1) to identify weak mask vectors, one would require prior knowledge of predictable codewords, which itself requires knowing the error vectors that are the most effective on the decoder function on-board according to Becker's suggestion in [67]. Alternatively, a relaxed version of this equation can coexist which is brought in (4.2). This alternative version suggests that if a candidate mask vector in XORing with a valid codeword from set C is equal to another valid codeword from the same set, then the mask vector is not effective against first order HDMA.

$$\forall c_i \in C, \exists m \Rightarrow m \oplus c_i = c_i' \; ; \; c_i' \in C \tag{4.2}$$

wherein c_i and c_i' are codeword elements of the main codeword set C. Using this alternative equation however, may come with the cost of reducing the chance of graduating a mask vector.

To experimentally assess the rate of mask vector rejection in our case, and to see how many candidate mask vectors are rejected according to (4.1), we took an statistical analyis over each of our SRAM-PUF dataset, considering each one at a time being a source of mask vector generation. Results of this statistical check are brought in Table 4.1. In comparison, we also brought the statistical analysis of mask vector rejection according to the relaxed equation (4.2), in Table 4.2.

Table 4.1: Number of rejected mask vectors out of 1600 per SRAM according to (4.1).

SRAM1	**SRAM2**	**SRAM3**	**SRAM4**	**SRAM5**
18	19	15	19	19

SRAM6	**SRAM7**	**SRAM8**	**SRAM9**	**SRAM10**
14	20	11	22	6

Table 4.2: Number of rejected mask vectors out of 1600 per SRAM according to (4.2).

SRAM1	**SRAM2**	**SRAM3**	**SRAM4**	**SRAM5**
608	608	480	608	608

SRAM6	**SRAM7**	**SRAM8**	**SRAM9**	**SRAM10**
448	640	384	704	192

4.3 Theoretical Analysis of masking with variable positioning

In theory, one can measure the min-entropy of the PUF responses w_1 and w_2 used for secret key generation and mask vector generation, respectively. Let us first discuss the measurement of uncertainty for an adversary w.r.t to correctly guessing the address of a given masked block within the masked primary helper data. We note that our formulation of the min-entropy is considered in a noise-free case.

Let us first denote the probability of masking a sub-division of a given helper data s as $P(s)$ with a given mask vector w_2. We can then define the entropy of the address to a masked sub-division for a given mask vector, H(Addr), as shown in (4.3).

$$H(Addr) = \sum_{i=1}^{x} P(s)_i \times (log_2 \frac{1}{P(s)_i}) \tag{4.3}$$

where x is the number of sub-divisions in the given helper data s that can be addressed for masking. Noting that in this formulation, we assume that the starting point of the address in the given mask vector w_2 is already known by the adversary.

Now we can consider a specification for $P(s)$, by defining the total number of sub-divisions in a given helper data s as $x = \frac{l}{k}$ where k is the size of a given mask vector and l is the size of the helper data vector. If we consider that $P(s)$ is equally distributed for all sub-divisions, we can then define the entropy of the address to a masked sub-division for a given mask vector as in (4.4).

$$H(Addr) = -\log_2 \frac{k}{l} \tag{4.4}$$

we can also develop (4.4) for cases in which two or more mask vectors are considered. Let us denote in masking with variable positioning, using m mask vectors, the length of the ith mask vector by k_i, and the length of the helper data vector by l. Noting that in this setting, we consider each mask vector out of the m blocks, having a variable size, thus denoting the length of the ith mask vector as k_i and not k. We would then have the entropy of masking address as in (4.5):

$$H(Addr) = -\log_2 \prod_{i=1}^{m} \frac{k_i}{l} \tag{4.5}$$

we can now use (4.5) in defining the min-entropy of the mask vector w_2 and the masking address. We would define it as $H_\infty(w_2, Addr)$ and refer to it as the min-entropy of masking.

$$H_\infty(w_2, Addr) = -(\log_2 \prod_{i=1}^{m} Max(P_r_\beta_i) + \log_2 \prod_{i=1}^{m} \frac{k_i}{l}) \tag{4.6}$$

where $Max(P_r_\beta_i)$ is the maximum probability of recovering one of the predictable codewords for the ith vector of the m mask vectors during the recovery phase.

Using equation (4.6) we can now compute the min-entropy of masking for the experimental cases we discussed in the previous section. Accordingly, the min-entropy for which we used 1 mask vector to mask a 256-bit helper data would increase by 5. This comprises the min-entropy of mask vector which is 1. Recalling that the maximum probability after error vector is applied to a [16,5,8] Reed Muller code (w.r.t the notation $[n, k, t]$, where n is the size of the codeword, k is the size of the binary random number and t is the order of the code) with majority logic vote decoder is $\frac{1}{2}$, and the min-entropy of the masking address is 4. Likewise for 2 mask vectors and 4 mask vectors to mask a 256-bit primary helper data, the min-entropy is increased by 10 and 20,

respectively.

We can also define the total min-entropy, which comprises the sum of the min-entropy of PUF response w_1 and min-entropy of masking. We would define it as $H_\infty(w_1, w_2, Addr)$:

$$H_\infty(w_1, w_2, Addr) = -(\log_2 \prod_{i=1}^{n} Max(P_r_\alpha_i) + \log_2 \prod_{i=1}^{m} Max(P_r_\beta_i) + \log_2 \prod_{i=1}^{m} \frac{k_i}{l}) \tag{4.7}$$

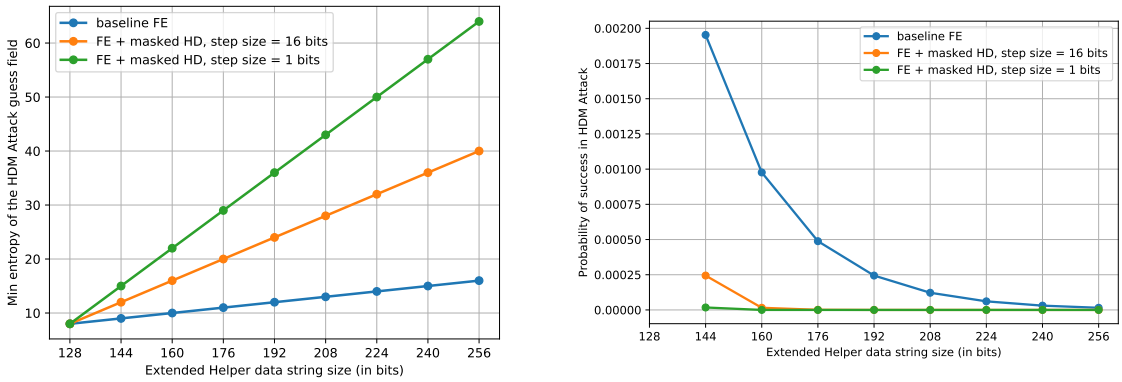
where $Max(P_r_\alpha_i)$ is the maximum probability of recovering one of the predictable codewords for the ith block of the n blocks of the primary SRAM-PUF response (i.e., the source of encryption key generation), during the recovery phase.

Noting that this specification of min-entropy is not bound to the limits of the primary specifications of our helper data masking with variable positioning scheme. One for instance can expand the address space of adaptive masking, but should however mind that it will then require more bits of the mask vector to define the address. In a noise-free setting, for S number of available maskable regions on primary helper data, the min-entropy of the HDMA against the key generation scheme would be:

$$H_\infty(w_1, w_2, Addr) = -(\log_2 \prod_{i=1}^{n} Max(Pr_\alpha_i) + \log_2 \prod_{i=1}^{m} Max(Pr_\beta_i) + \log_2 (\frac{1}{S})^m) \tag{4.8}$$

To theoretically analyze the efficiency of our helper data masking, we compare 3 approaches in strengthening the security of an FE based key generation scheme. At one approach, the w_1 string is extended without considering to mask the helper data. At another approach, the w_2 string which is the mask value, is extended and the positioning step is defined to be bit-wise. Therefore, the number of possible positions of the mask over the helper data string for w_1 is $n - m + 1$, where n is the size of w_1 and and m is the size of w_2. The other approach here is to consider for each added extension block for w_2, the positioning step to be the size of the added block. Therefore, the total number of possible positions of the mask over helper data s would be $\frac{\text{size of}(w_1)}{\text{size of}(w_2)}$.

Given equation (4.8), we can demonstrate the increase in the min entropy of beckers HDM Attack against a $[16, 5, 8]$ Reed Muller code and a majority logic vote (MLV) decoder. Initially when encoding, the number of possible codewords are 32 given the



(**a**) (**b**)

Figure 4.5: Illustrations showing (a) Min entropy of guessing in Becker's HDM Attack against [16,5,8] Reed Muller Code and Majority Logic Vote decoder. (b) Probability of success of Becker's HDM Attack against [16,5,8] Reed Muller Code and Majority Logic Vote decoder.

specification above. However, as discussed in [67], HDM Attack can reduce it to 2 possible codewords after modifying the helepr data.

We consider the initial size of the key to be 128 bits. Figure 4.5a shows the increase in the min entropy for the three approaches with respect to extending the helper data string. It is apparent that the ' FE + masked Helper Data' with 1 bit positioning step size has the lead compared to the other two. The ' FE + masked Helper data' with 16-bits positioning step also has a significant lead compared to the baseline FEs. Recalling that in baseline FE we just increase the size of the w_1 sting which is the source value for generating the encryption key.

We can also assess the probability of success of HDM Attack. The probability of success can also be defined as in (4.5b).

$$Pr(w_1, w_2, Addr) = \prod_{i=1}^{n} Max(Pr_\alpha_i) \times \prod_{i=1}^{m} Max(Pr_\beta_i) \times \prod_{i=1}^{m} \frac{k_i}{l} \tag{4.9}$$

where $Max(P_r_\alpha_i)$ is the maximum probability of recovering one of the predictable codewords for the ith block of the n blocks of the primary SRAM-PUF response. And $Max(P_r_\beta_i)$ is the maximum probability of recovering one of the predictable codewords for the ith vector of the m mask vectors during the recovery phase. In Figure 4.5b, we

Table 4.3: Min entropy of various codes and decoding methods. The two different codes mentioned here are Hard-in Soft-out.

Code	Decoder	Reliability	Min-entropy (per codeword) No Masking	Min-entropy (per 175 bits) No Masking	**Min-entropy (per all bits) 1 block mask 5-bit step size**	**Min-entropy (per all bits) 1 block mask 1-bit step size**	**Min-entropy (per all bits) 2 blocks mask 1-bit step size**
[7,1,7] Repetition Code [16,5,8] Reed-Muller Code	SDML Soft-decision [73]	100%	1	35	**40.09**	**42.38**	**49.66**
		95%	4	138.8	**143.89**	**146.18**	**153.46**
	SDML hard-decision [73]	100%	1	35	**40.09**	**42.38**	**49.66**
		95%	1.5	51.8	**56.89**	**59.18**	**66.46**
	Without Attack		5	175	**180.09**	**182.38**	**189.66**
	GMC [74], [75]	100%	0	0	**5.09**	**7.38**	**14.66**
		95%	2.4	82	**87.09**	**89.38**	**96.66**
	Without Attack		5	175	**180.09**	**182.38**	**189.66**
	Majority logic[67]	100%	0.2	6.8	**11.89**	**14.18**	**21.46**
		85%	0.2	11	**16.09**	**18.38**	**25.66**
	Without Attack		5	175	**180.09**	**182.38**	**189.66**
[8,1,8] Repetition Code [24,12,8] Golay Code	Soft Decision [76]	100%	0	0	**3.32**	**6.77**	**13.2**
		95%	4.1	44.9	**48.22**	**51.67**	**58.1**
	Without Attack		12	132	**135.32**	**138.77**	**145.2**

can observe the decreasing probability of success for the HDM Attack with respect to extending the helper data string. It can be seen that overall the ' FE + masked Helper Data' with 1 bit positioning step size is the strongest countermeasure. Nonetheless, ' FE + masked Helper Data' with 16 bits positioning step size has a similar characteristic. Nonetheless, both masking approaches seem to be considerably stronger compared to the baseline FE. This suggests ultimately that masking helper data using PUF data seem to be a reliable and strong countermeasure. Thus at the end that considering to use a part of w_1 to construct a w_2 mask value and mask the helper data, can lead to a stronger FE based key generation scheme. In the following, we demonstrate our evaluation of HDM Attack against some real SRAM PUF data we collected from an in-house developed SRAM device.

4.3.1 Evaluation on various Coding and Decoding methods

Reed Muller (RM) codes have been used in abundance for error correction codes in several applications where biometric data is used to generate secret values. Some of the use cases of RM codes are mentioned in [73]–[77]. An evaluation of HDM Attack against RM codes is presented by Becker in [67] where the the codes are used to recover original secret values generated from PUF source. This work shows how the new HDM Attack can decrease the min-entropy of the recovering codeword by injecting an error vector e into the decoding process. We restate his analysis of HDM Attack over various codes and decoding methods in TABLE.4.3 and demonstrate as well the effect of using our masking countermeasure on the entropy of the recovering codeword. Here the re-

sults show that using our helper data masking masking mechanism overall increases the entropy of the codeword. One can observe here that the entropy of codeword after using HDM Attack on decoders such as on SDML soft-decision, where the reliability is not 100%, is considerably high. Therefore the use-case of our masking mechanism, although it adds to the overall entropy, may not be justified for such cases.

On the other hand, the increase in entropy is relatively high using our helper data masking mechanism for cases such as Soft Decision Hackett at 100% reliability, Majority Decoding at both 100% and 85% reliability and GMC decoder at 100% reliability. We can assume here that masking helper data can in turn be accounted as an effective countermeasure against HDM Attack where the overall entropy of the codes are low. Given that so far we only considered maximum 2 block of the entire bits to be the mask values. In turn, if the overall overhead of computing large number of masking is justified (i.e., considering half of the bits as the stem value for key generation, and the other half for masking), we can count on the linear addition they provide on the overall entropy of the codewords against HDM Attacks.

4.4 Experimental Setup

4.4.1 Description of SRAM PUF device

The SRAM-PUF device we developed is an in-house device with a software package allowing us to interact with and control the device. Figure 4.6 shows the device housing the micro-controller with our SRAM-PUF. Some of the functions implemented as part of our SRAM-PUF include a function for reading the real time PUF response after refreshing the SRAM every 2 seconds, and a function for enrolling the PUF, where it performs 100 read-out from the PUF source in the SRAM device and stores the responses.

Our SRAM-PUF follows the addressable PUF generator (APG) technique, which is described in [78]. This allows us to extract unique responses from the PUF-device. The APG mechanism randomly selects memory cells to feed the PUF. For each memory address queried to the PUF device, there will be several response values coming from the memory cells. After an arbitrary amount of acquisition, the memory addresses and the captured responses are stored into a PUF dataset.

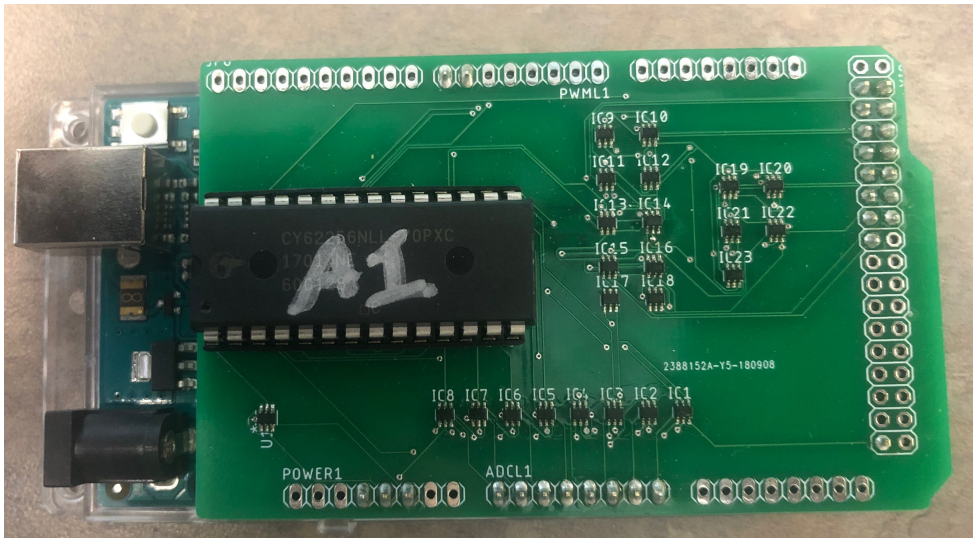


Figure 4.6: Our in-house developed SRAM device.

4.4.2 PUF Data Description

We employed 10 SRAM-PUF devices for evaluation. For a given SRAM-PUF device, at the same memory address, we read the responses from each memory address 100 times. Noting that each memory address yields a 256-bit binary vector as the response. Ideally, the captured 100 response vectors should be exactly the same since they belong to the same address. However, due to the process variation between each response acquisition, there will be a chance that each cell's binary value would flip. Therefore, the PUF response vectors in-between different acquisitions have dissimilarities in some cell values. This implies the inherent physical characteristic of a real PUF.

Here we assess $\{16, 32, ..., 128,$ and $256\}$-bit of PUF responses. Noting a PUF response smaller than 256-bit, is a sub part of the 256-bit PUF response we captured initially from SRAM-PUF instances. For instance, a 16-bit PUF response is the first block of the captured 256-bit PUF response. Similarly, a 32-bit PUF response is the first two blocks of the captured 256-bit PUF response and so on.

In addition, we also measured the frequency of bit-flipping within each subset, to identify the unstable cells corresponding to each PUF images. Following Table 4.4 at the end of the paper, it shows the percentage of cells with bit flipping per block of 16-bits within the entire 256-bits of a PUF image for a given memory address, per SRAM. Nothing that this statistical analysis is performed on the 100 captured PUF images from each SRAM for a given memroy address.

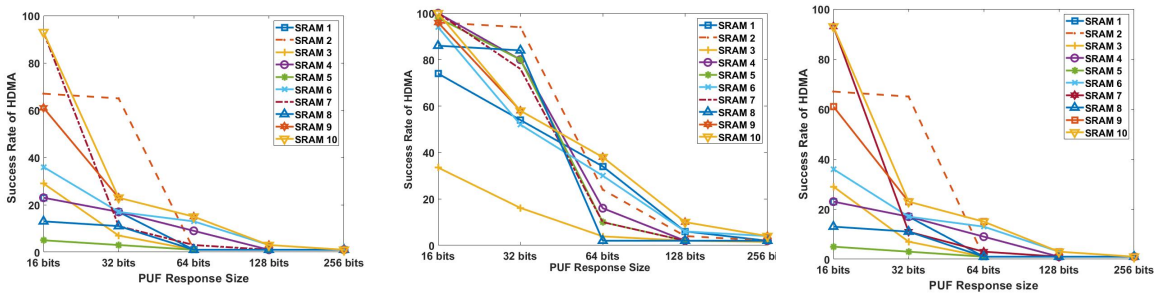
Table 4.4: Percentage of unstable cells, per block of 16-bits, in a PUF response size of 256 bits per SRAM

	SRAM1	SRAM2	SRAM3	SRAM4	SRAM5	SRAM6	SRAM7	SRAM8	SRAM9	SRAM10
Block1	18.75%	12.50%	18.75%	12.50%	18.75%	12.50%	6.25%	12.50%	12.50%	12.50%
Block2	18.75%	6.25%	18.75%	25.00%	18.75%	12.50%	25.00%	6.25%	18.75%	18.75%
Block3	12.50%	18.75%	6.25%	6.25%	12.50%	12.50%	37.50%	37.50%	6.25%	6.25%
Block4	18.75%	12.50%	18.75%	18.75%	25.00%	18.75%	31.25%	6.25%	18.75%	18.75%
Block5	12.50%	25.00%	12.50%	18.75%	12.50%	0%	31.25%	25.00%	6.25%	6.25%
Block6	0%	18.75%	6.25%	6.25%	31.25%	12.50%	0%	0%	6.25%	6.25%
Block7	12.50%	18.75%	25.00%	12.50%	18.75%	25.00%	6.25%	25.00%	25.00%	25.00%
Block8	18.75%	18.75%	6.25%	18.75%	25.00%	6.25%	18.75%	37.50%	6.25%	6.25%
Block9	25.00%	6.25%	18.75%	25.00%	25.00%	18.75%	31.25%	25.00%	6.25%	6.25%
Block10	0%	12.50%	12.50%	18.75%	18.75%	0%	6.25%	12.50%	12.50%	12.50%
Block11	25.00%	12.50%	31.25%	37.50%	18.75%	25.00%	12.50%	6.25%	12.50%	12.50%
Block12	12.50%	25.00%	12.50%	18.75%	12.50%	18.75%	12.50%	12.50%	0%	0%
Block13	0%	12.50%	18.75%	18.75%	18.75%	6.25%	18.75%	25.00%	0%	0%
Block14	31.25%	25.00%	18.75%	6.25%	6.25%	12.50%	12.50%	12.50%	6.25%	6.25%
Block15	25.00%	18.75%	6.25%	18.75%	18.75%	18.75%	18.75%	31.25%	12.50%	12.50%
Block16	18.75%	18.75%	18.75%	18.75%	18.75%	37.50%	12.50%	25.00%	0%	0%

4.5 Experimental results

In this section, the results of performing HDMA against our SRAM-PUF instances datasets will be discussed. Figure 4.7 shows the best case, the worst case and the average success-rate of HDMA, respectively. We clarify first that the difference in the attack success-rate between the best case and the worst case in plots (a) and (b) in Figure 4.7 is due to the difference in the value of the codeword used per SRAM-PUF dataset per given PUF response. Recalling that the possible number of codewords are 32, due to using a 5-bit binary random number for encoding. We can see that HDM Attack against the employed Reed Muller code and the majority logic vote decoder has some significant chance of success if the PUF response is small. Although we can see that at 128-bits size, there still is a chance of success, even in the best case scenario. At 256-bits obviously due to the doubled entropy of the guess field, no success was observed in the HDM Attack at 100 attempts per device.

We now observe the success-rate of HDM Attack against FE-based key encryption with helper data masking with variable positioning. To define one source for key value and one for mask value generation, we define here 5 virtual devices wherein each device employs 2 SRAMs. Thus, each virtual device is defined to comprise 2 SRAM datasets according to Table 4.5. Accordingly, there is a primary SRAM dataset that is the source of key generation and the secondary SRAM dataset for mask vector generation. We note also that we employed the variable positioning step size of 16-bits which is equal to the



(**a**) Best case success-rate (**b**) Worst case success-rate (**c**) Average success-rate

Figure 4.7: Success-rate of Becker's HDMA out of 100 attempts on 10 different SRAMs

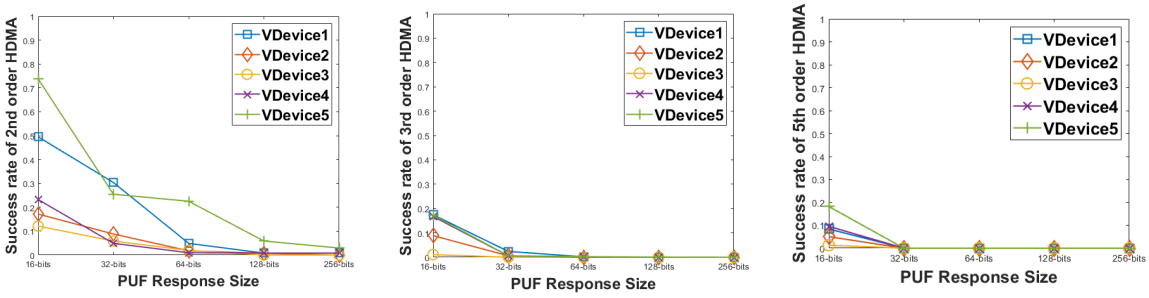
Table 4.5: Definition of virtual devices.

Virtual device	**SRAM for key Generation**	**SRAM for mask vector generation**
Vdevice1	SRAM1	SRAM2
Vdevice2	SRAM3	SRAM4
Vdevice3	SRAM5	SRAM6
Vdevice4	SRAM7	SRAM8
Vdevice5	SRAM9	SRAM10

size of a codeword in this work.

This experiment is performed in three settings wherein each setting, different sizes of mask vector is considered. Figure 4.8 shows the results of this experiment in settings of using 1-block mask vector, 2-block mask vector, and 4-block mask vector. The plots shown in the figure represent the decreasing success-rate of second order HDM Attack against 5 virtual devices as the size of PUF response for key generation increases. Noting that for the assessment using 1-block mask vector, the datasets of PUF responses for key generation are augmented from 100 responses per dataset to 10000 responses. This augmentation is performed by simply repeating each response 100 times. For the assessment using 2-block mask vector and 4-block mask vector also, the datasets are augmented to 100000 responses per dataset using the same augmentation method. Recalling also that the responses per dataset correspond to one address of the SRAM-PUF and the differentiation between them is due to the instability of the SRAM-PUF.

First, we point out to the case where the success-rate of the attack against a PUF response size of 16 bits is still high regardless of the masking. We expect it since for a 16 bits of helper data, there is only one addressable block to mask. Therefore, there is no entropy in the guessing field for the address space on the adversary's side. Other than



(**a**) 1 × 16-bits mask value (**b**) 2 × 16-bits mask value (**c**) 4 × 16-bits mask value

Figure 4.8: Success-rate of HDM Attack with a) 10000 attempts. b) 100000 attempts. c) 100000 attempts.

that, we can see that using masking with variable positioning significantly decreases the success-rate of second order HDM Attack while the PUF response is larger than the mask block. Unlike the key generation without masking, the success-rate drops drastically at even smaller w_1 sizes such as 32-bits and 64-bits while we employ more than 1 mask vector. This is similar to the demonstration of probability of success as we discussed in Section IV. Noting in the case of the settings using 2-block and 4-block mask vector, the success-rate has been measured from 100000 attempts of second order and fourth order HDMA, of which the results are shown in Figure 4.8(b) and Figure 4.8(c), respectively.

4.6 Related Works and Comparison

Our proposed helper data masking is not the first work to provide a countermeasure against HDM attacks. Several countermeasures have already been proposed in [79]–[81]. Here we discuss some of the existing countermeasures and compare them qualitatively with our method.

The work of Delvaux et al. in [80] explains four schemes that secure a Helper Data Algorithm (HDA)-based key generation process. The primary HDM attack detection method is originally proposed in [82]. The method suggests to provide a hash value using collectively the helper data and the key value on the server side, and publicize it for devices to query during key generation. During key generation, the device queries the server for the hash value and compares it with a locally generated version of the hash. If equal, the generated key is considered valid and the key can be used for encryption and decryption of the exchanged message. We have shown a similar key generation struc-

ture as shown in Figure 4.1 which is similar to the RFE-like process discussed in [67]. The problem with this scheme is that an HDM attacker can impersonate the server and provide the hash value itself, which is based on the manipulated helper data and the guessed key value. Therefore the proposed countermeasure can be easily compromised. Considering our countermeasure however, we showed that the demasking process cannot be compromised by any modification since the position of the mask over the helper data vector is based on the true value of the mask.

Similar to [82], another HDM detection scheme has been proposed [83], in which only the key value is used to generate the hash. However, the same security issue as we explained above can be discussed here. The HDM detection can be biased just by the attacker impersonating the server and providing its own hash value for key validity check on the PUF-enabled device.

Another solution has been discussed in [80] where it is suggested to program the public key and the helper data all into a one-time programmable NVM memory (EEPROM) during the enrollment phase of the PUF-enabled device [18]. During the key regeneration then, the IC housing or working with a PUF component for key generation, queries the NVM memory to retrieve the helper data for key generation. This solution ultimately negates possibility of HDM attack as we discussed in this work, since there is no out of device communication with a server system to query for helper data. One can assume however that device can undergo some physical attacks using side channel analysis methods to read the information on the NVM device, and also using fault injection attacks to change some bits in order to impose helper data manipulation. These attacks however need precise tools to perform such modifications from outside of the device. This means that the attackers are enforced to employ some expensive methods. Nonetheless, the countermeasure itself is an expensive implementation, since it requires an external device to store the public key and the helper data. Moreover, using a memory component accompanying a PUF is fundamentally questioning the employment of PUF itself. Since the existence of PUF is set to replace conventional methods which save the secret values on device memories in order to provide protection against physical attacks.

A countermeasure against HDM attack is proposed by Hiller et al. in [84]and discussed also in [80]. The problem discussed in their work is the practice of HDM attacks that focus on single codeword modification by observing several key generation operation outputs. The authors then propose a scheme where a hash value of the helper data

is XORed with the recovered PUF response to generate the key. This way, the authors ensure that at list 50% of the key values are changed and not a single one, which in turn disturbs the entire attack mechanism. The issue with this countermeasure is that it is targeted for attack methods which aim to recover the original key. While there exists other HDM attacks such as the one discussed by Becker in [67], for which this countermeasure is equally vulnerable to that of the RFE-like method we discussed in this work.

As we explained earlier, the mechanism of robust fuzzy extractors (RFE) can also provide some level of security. This has been discussed already in [67] and [82]. In that mechanism, despite the equality check between a localy generated and a publicly received hash value on the PUF device, the hamming distance between the recovered PUF response and the raw extracted PUF response is measured as well. If the the distance is larger than a threshold, then the key generation fails. This suggests that only recovered codes with certain Hamming distance from the original ones are valid when recovered. However, such mechanism can also discard regenerated keys which have not been affected by an HDM attack. Simply due to PUF instability, there could be more noisy bits than what the RFE's Hamming distance threshold allows for a valid recovered codeword. Therefore, this method, although providing security against HDM attacks, also imposes extensive sensitivity to PUF instability.

Gao et al. in [79] proposes a PUF-based key generation technique and assures its security using BCH codes and syndrome decoding. Using BCH codes and syndrome decoding has good level of security against HDM Attacks as discussed in [67]. However, it is also discussed that for $[n,k,2t+1]$ BCH codes with small k, it is not as efficient to use syndrome decoding compared to other simpler decoders such as maximum likelihood decoding.

Merli et al. in [81] proposes a codeword masking scheme against helper data manipulation attacks based differential power analysis (DPA). The DPA HDM attack in their work aims to read the processed output of an inner decoder which decodes the noisy codeword that is the product of the XOR of the noisy PUF response and the received helper data. The codeword masking countermeasure in turn is implemented in a way to obfuscate the output of the inner code, so that DPA is not capable of reading the raw output of the first stage decoder. In order to mask the inner code, a random locally generated mask value is first encoded and the encoded mask value is XORed with the helper data, which is then XORed with the noisy PUF response. The output

is then decoded and then demasked using the raw mask value to yield the secret key value. The countermeasure proposed here is the closest to our work. The similarity is that by masking the inner code using a random value, the DPA cannot find the correlation between the captured power traces. Theoretically, when masking is applied to secure an implementation, the DPA will require considerably more traces in order to bypass the masking and find the correlation to extract the target value. This is similar to increasing the guessing entropy of the HDM attack as we discussed in this work. The increased entropy in turn requires more observations on the data transmitting channel to discover the correct guess for the location of the mask on the helper data.

Most of the countermeasures discussed here were methodical and aimed at preventing the HDM attack using hardware or algorithmic solutions. Few, such as [81] suggested methods focus on the theoretical aspect of HDM attack and provide solutions that make the attacks more difficult to succeed. Our method also sits in this class of countermeasures. Since we also try in general to increase the guess entropy of the attacker using the proposed helper data masking method.

As was apparent however, in this countermeasure we did not practice or incorporate the potentiality of ML solutions. In the next chapter however, our goal is to discuss that and propose a novel authentication and key exchange protocol that utilizes ML model of PUF. There we elaborate on a repetition-like coding technique that is locally recoverable using the two identical sources of the CRPs, one is the hardware PUF and the other is the predictive model of the PUF which we make during the enrollment of the PUF on the server side. We show that we can make such protocol ultimately secure by exchanging only challenge values. Such communication schematic as we elaborate on can be considered inherently secure against man in the middle attacks. Then we also show that the protocol can be adaptive towards different level of noisiness and keep the success-rate of recovering the secret value at 1.0. In the last chapter also, we elaborate on a technique to lower the size of exchanged data and also obfuscate the challenge values as well as a preventive security solution against future attacks that may be interested in the publicly exchanged challenge values as well.

4.7 Designing a Novel Secure Key Exchange Technique using ML and Strong PUF

In this work, we propose a new encoding and decoding method for strong PUF-based key generation and exchange, which is capable of recovering an original key using no public helper data information. Here we propose to use a strong PUF with increased complexity, such as XOR Arbiter PUF with an XOR size of larger than 3. We also propose using PUF with input size larger than 64 bits. This increases the size of the CRP space and will enable generating redundant codes locally which in turn increases the probability of successfully exchanging the encryption keys only between the trusted parties. In order to provide the capability of generating a large number of randomized encryption keys, we propose using a machine learning-based equivalent model of the PUF on a Trusted Third Party (TTP) verifying server. The ML model of PUF in turn gives access to its full CRP space with some negligible error probability. This will enable us to generate one-time usable key values which is a secure mechanism against replay attacks. Here we show how our method increases the probability of success in exchanging encryption keys using challenge packs that generate mutual response values, a mechanism similar to repetition coding. In such mechanism only, a series of randomized challenge vectors are transmitted that have no correlation to the response values that are the source for generating the encryption keys. Our method can be used for one-to-one and one-to-many (multi-party) communications. Our contribution is listed as below:

- A repetition code-like error correction method using the PUF and an ML model of the PUF.
- A one-to-one and one-to-many key exchange protocol with locally correctable codes.
- A novel authentication technique between the TTP and the PUF-enabled device.
- An evaluation of our proposed method using simulated data of various XOR Arbiter PUF model structures with different levels of noisy CRPs.

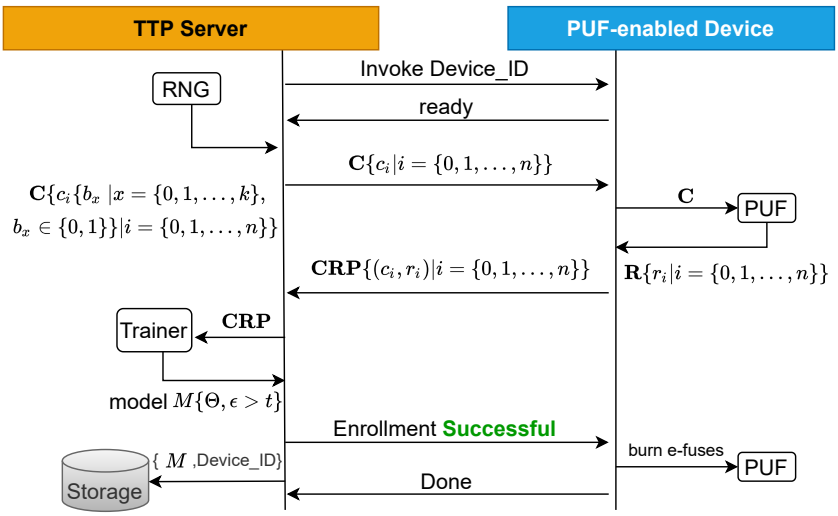


Figure 4.9: Illustration of the enrollment procedure.

4.7.1 Proposed Method

The main goal of our proposed method is to provide a medium for a secure encryption key generation and exchange between two or more trusted parties. We define three phases that constitute our method, 1) The enrollment phase, 2) The challenge-based synchronization phase, and 3) The authentication and key generation and exchange phase.

The process of enrolling a PUF-enabled device to a TTP server is illustrated in Figure 4.9. Only during the enrollment phase, we assume that the communication channel between the TTP server and the device is private and only the authorized parties can audit the channel. We also assume that the access to the PUF is facilitated initially with E-Fuses which are later to be burned/disabled after the device is successfully enrolled. In the enrollment phase, the TTP server first invokes the device to prepare it for the CRP read-out. The TTP server then generates n number of randomized challenge vectors (e.g., a binary vector of size k which is equal to the number of stages in the corresponding PUF) to create a challenge set C, and sends C to the device. The PUF-enabled device then acquires n number of responses into a set R and sends the CRP set back to the server. The TTP server uses the CRP set to train a predictive model M with minimum prediction accuracy $\epsilon >$ t. After successfully training the model, the TTP server saves the model with the ID of the enrolling device and sends feedback to the device to burn/disable the E-Fuses.

The challenge-based synchronization phase is shown in Figure 4.10. This is the ini-

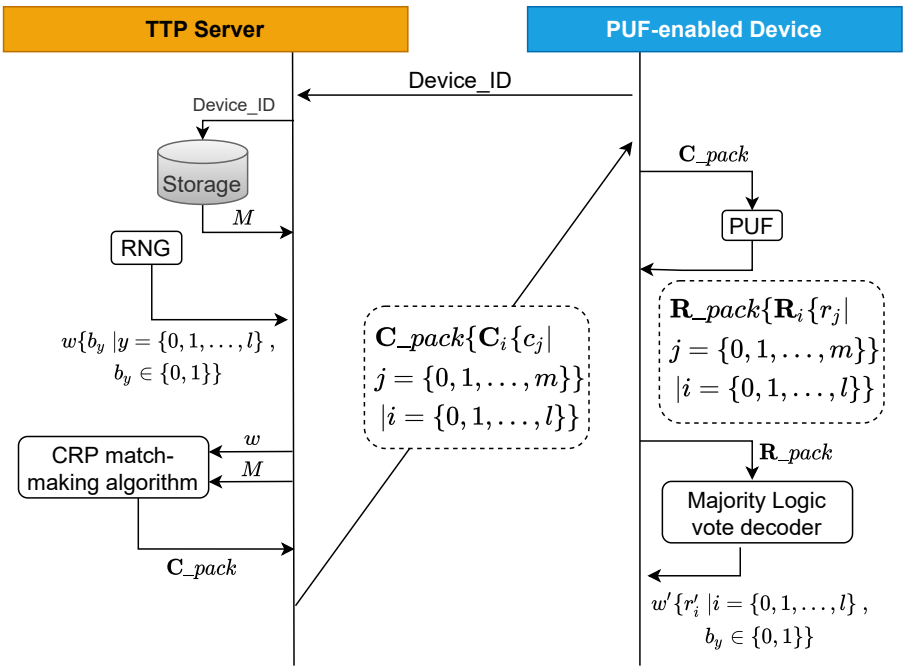


Figure 4.10: Illustration of the challenge -based synchronization.

tial phase when the PUF-enabled device demands a new encryption key. Thus, at first, the device queries the TTP server by exchanging its Device_ID. Once received by the TTP server, the corresponding predictive model of the device's PUF is loaded. Then the server generates a random l-bit binary vector w using a random number generator (RNG). The generated random value w and the predictive model M are given to a CRP matchmaking algorithm.

The CRP matchmaking algorithm as shown in Algorithm (1) is responsible for generating a set of randomized challenge vectors we refer to asC_pack, which includes l number of Ci subsets comprising m number of challenges that produce the same response. The model M here gives the response to any given challenge vector cj. The CRP matchmaking in turn acts similarly as a repetition code. While the output is not directly the codeword, it is instead the challenge vectors that will lead to the binary response values which then constitute the codeword. This will then allow to securely enable regeneration of the secret value on the device side. One can also assume the PUF and the model M as the two ends of a noisy channel where the repetition codeword is being transferred through, as if the predictive model is the channel source and the actual PUF on the device side is the channel destination, and the PUF instability and model mis-prediction error are the sources of noise for the channel.

The generated C_pack set is then sent to the device. On the device side, C_pack is passed to the PUF to produce the R_pack set, which comprises l subsets, and each subset comprises m binary response values. Once the R_pack is extracted, each of the subsets is given to a majority voter to vote for the most dominant binary value. The output of the voter for the entire R_pack is w' which comprises l binary values. Here it is expected that w' is equal to w, which is the base condition to issue a successful device authentication and encryption key exchange.

Algorithm 1 CRP matchmaking algorithm

Require: $w\{b_y|y = \{0, 1, ..., l\}, b_y \in \{0, 1\}\}$
Require: $M\{\Theta, \epsilon > t\}$
  $L \leftarrow l$ ▷ size of w
  $M \leftarrow m$ ▷ repetition length
  while $i < L$ **do**
    while $j < M$ **do**
      $ch \leftarrow$ randomly generate challenge vector
      $rp \leftarrow M(ch)$
      if $rp = w[i]$ **then**
        if $ch \notin C_pack$ **then**
          $j \leftarrow j + 1$
          $C_pack[i] \xleftarrow{+} ch$ ▷ Append ch to C_pack1
        end if
      end if
    end while
    $i \leftarrow i + 1$
  end while
  return C_pack

We assume that a request for synchronization initiates a new session. Once completed, the communicating parties can initiate device authentication and key exchange. Figure 4.11 shows the process of device authentication and key exchange between the TTP server and the PUF-enabled device. For device authentication, the TTP server first initiates the request to the PUF-enabled device. The request triggers the device to create a hash value $h1'$ using its own $Device_ID$ and the generated w' from the synchronization phase. TTP server also generates a hash value $h1$ using the generated secret value w from the synchronization phase and the communicating device's $Device_ID$. The PUF-enabled device then sends $h1'$ to TTP server. On TTP server, if $h1$ and $h1'$ are equal, it is assumed the communicating device is authentic and the TTP server sends the authenti-

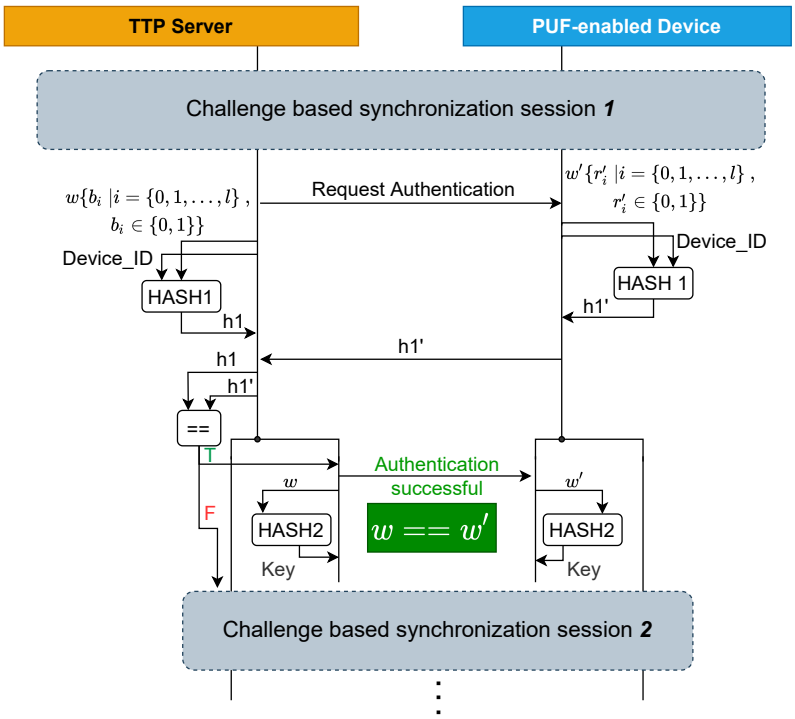


Figure 4.11: Illustration showing the authentication and key generation and exchange procedure part of our proposed method.

cation successful acknowledgement to the device, meaning that it is known to both parties that their generation w and w' are equal. Both parties then move to the creation of the encryption key. The TTP server and the device create the Key value which is a hash value generated using the w and w', respectively. While no information is exchanged which correlate to the secret value w and w', the entire process from synchronization to the key generation and exchange phase assures that both values are equal, using the repetition code-like mechanism we proposed.

4.7.2 Related Works

The repetition code-like error correction and the utilization of a predictive model of PUF for authentication and key exchange are of the core novelties of our work. Nonetheless there are several works that practice similar mechanisms for their PUF based key generation/exchange and authentication. Below we discuss few of these works.

Slender PUF is an authentication protocol proposed by Majzoobi et al. in [48]. In this protocol, the authors propose a mechanism of CRP exchange where the CRP on the server side is generated from a pseudo random number generator (PRNG) coupled

with a compact model of the PUF that enables the user to generate CRPs randomly with no restriction in the number of CRPs. Slender PUF protocol is one of the primary attempts that suggest leveraging the modeling of PUF with machine learning to build a robust authentication mechanism. However, in their protocol, they also simply disclose the CRPs on public communication channels during authentication operations, which ultimately renders the mechanism prone to model-building attacks.

Idriss et al. proposed a lightweight highly secure PUF based device authentication method in [85]. This method is one of the few authentication methods that is based on only exchanging challenge values and disclose no response value during an authentication operation. Here the primary step is that the device sends it's ID to the server and later sends a $C_packci|i = 1, 2, ..., l$ to the server. Given that for the C_pack there is a designated $R_packri|i = 1, 2, ..., l$ where $ri = PUF(ci)$. The server then receives the C_pack and generates a $C_pack'(ci, 1, ci, 2)|i = 1, 2, ..., l$ where $P\bar{U}F(ci, 1)XORP\bar{U}F(ci, 2) = ri$, and $P\bar{U}F$ represents the lookup table of the stored CRPs of the corresponding PUF on the server. This method ultimately secures the authentication against model building attacks since no response value is exposed to the public. Nonetheless, since they propose using a CRP lookup table on the server side rather than a model of the PUF, then the protocol will ultimately face a limit in terms of the number of CRPs it can use.

Quadir et al. present a novel key generation mechanism based on strong PUF in [50]. In their mechanism they use a predictive model of the PUF on the server side. This protocol also proposes exchanging only challenge values. Thus, once a C_pack is exchanged between the server and the device, both are able to generate an R_pack and R_pack' respectively, which have logical differences due to PUF instability and PUF model prediction error. The R_pack and R_pack' sets are source to generating the encryption key, however their differences need to be corrected so that both response sets are expected equal. To do so, Quadir et al. proposes using helper data-based fuzzy extractors. While the helper data-based fuzzy extractor can guarantee the robustness of the encryption key, it is however prone to helper data manipulation attacks as we explained earlier in this paper.

4.7.3 Evaluation of Reliablity

To evaluate the reliability of our method, we used a Python-based Arbiter PUF simulator to generate 10 instances of each 4,5,6, and 7 XOR arbiter PUF variations with a 64-bit challenge size. The simulator code is developed by Ruhrmair et al. and presented

Parameter	**Value**
Optimization function	Adam
Loss function	BCEloss
Learning rate	0.001, 0.0001
Weight initializer	Kaiming Uniform
Bias initializer	Unifrom
Maximum epoch	2000
Maximum re-training attempts for Transfer Learning disabled	10
Maximum re-training attempts for Transfer Learning enabled	2
Training batch size	= Training set size
Optimal Test Accuracy	90%

Table 4.6: Training specifications & Hyper-parameters

initially in [26] and is available online in [51]. To represent a realistic characteristic, we added artificial bit flipping characteristic to the captured CRP datasets from the simulated PUF instances. Overall, we considered different noise levels, including 0%, 2%, 5% and 10% of CRP population for each dataset to be affected by bit flipping. We assessed the modeling of the PUF instances, and the performance of our proposed method separately for each noise level.

Using Pytorch, we created the corresponding predictive models of the PUF instances using a novel Multi-Layer Perceptron (MLP) proposed by Mursi et al. in [6]. We implemented the training procedure as proposed in [86] and we used a transfer learning technique as proposed in [87] as well to reduce the number of CRPs required for training an accurate model. The MLP model structure and the training parameters are given in Table 4.6.

We measured the accuracy of the models we trained for each PUF instance. Ideally we consider a model successfully trained if it has prediction accuracy above 80%. Table 4.7 shows the overall results of the models we provided from the simulated PUF instances.

To measure the performance of our key exchange method, we implemented the enrollment phase, the challenge-based synchronization phase and the key generation and exchange phase on python, by developing the class of a TTP server and a PUF-enabled device. We also implemented the exchange mechanism as an interface between the two classes, and a measurement procedure to measure the success-rate in terms of num-

Table 4.7: Training performance

XOR size	noise level	N0. CRPs	success-rate	model acc (ϵ)
4-XOR	0%	40k	1.0	99%
	2%	100k	1.0	97%
	5%	100k	1.0	94%
	10%	100k	1.0	89%
5-XOR	0%	90k	1.0	99%
	2%	200k	1.0	97%
	5%	200k	0.8	94%
	10%	200k	1.0	90%
6-XOR	0%	500k	0.5	98%
	2%	700k	0.9	97%
	5%	700k	0.4	94%
	10%	700k	0.3	90%
7-XOR	0%	950k	0.9	99%
	2%	950k	0.2	97%
	5%	950k	0.5	94%
	10%	950k	0.5	90%

ber of times that both the TTP server and the PUF-enabled device obtain a mutual key successfully. We measure the success-rate of key generation for each noise level, with respect to increasing repetition length l, testing the key generation also for 1000 iterations for each case. Figure 4.12 shows the success-rate progress with respect to increasing repetition length for various XOR sizes. As expected, we can see that for all XOR sizes, as the noise levels increase, the success rate degrades relatively. Given however, that with increasing l, the success-rate increases as well to compensate for both the noisy PUF source as well as the mis-prediction error-rate in the equivalent MLP model. We can see that for the worst case of having 10% of noisiness in the PUF characteristic, as well 10% mis-prediction error-rate, with repetition length above 10, we can achieve a success-rate in key exchange reaching and stabilizing at 1.0.

4.7.4 Security Analysis

Below we also evaluate the security of our method for various known attacks.

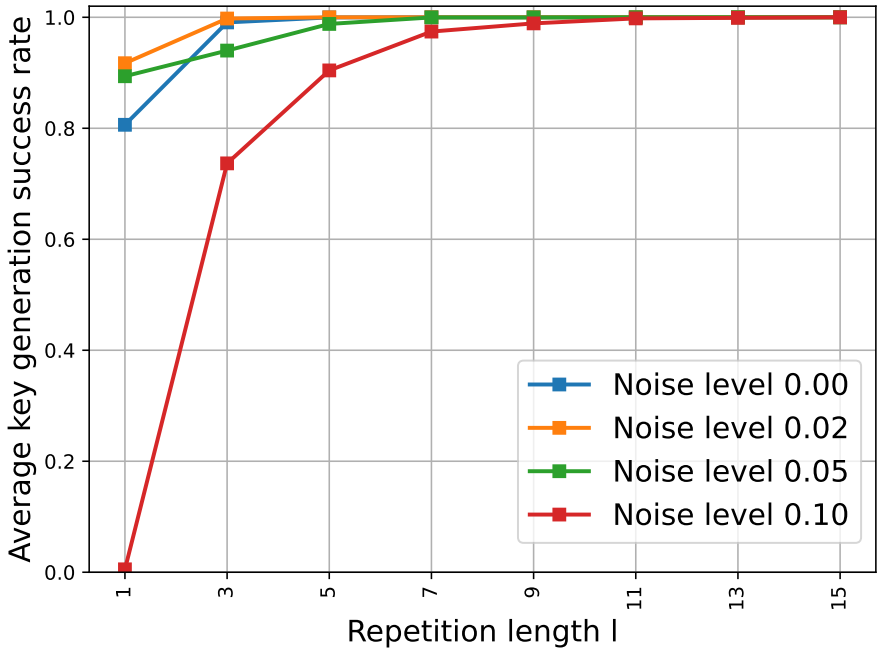


Figure 4.12: Key exchange success rate with respect to repetition code length l.

Machine Learning Modeling Attacks

Modeling attacks on strong PUF require eavesdropping on the communicating channel to capture PUF CRPs which is a necessity to build a model of a strong PUF[5], [7], [26], [54]. It is proven that with enough number of CRPs, relative to the complexity of the structure of the Strong PUF, it is possible to model any PUF using machine learning methods. If an attacker successfully models a PUF, he will be able to impersonate the target device and query the server for important information and be able to decrypt them using the model of the PUF. In our key exchange mechanism however, this way of attacking PUF is not possible since only randomly generated challenge values are exchanged, and without the response value for each challenge, modeling the PUF is not possible.

Helper Data Manipulation Attacks

We discussed earlier that PUF-based key exchange and key generation protocols need error correction codes to build robust encryption keys. Mainly the fuzzy extractors using publicly available helper data are employed in PUF-based key generation methods to recover the original key value from the noisy PUF CRP [88]. However, availing helper

data publicly has shown to be a weak spot for such methods. There exist adversary methods that manipulate the helper data and redirect it to the target device with the aim to bias the underlying recovering codeword during the decoding phase and lead the final value to a subset of guessable values by the attacker [67]. This vulnerability is mainly existing due to the publicity of the helper data. We showed here however, that it would be possible to recreate the codewords using the PUF itself and only publicize challenge values which are randomized and have no correlation to the secret value without any knowledge of (e.g, the predictive model of the PUF) or physical access to the PUF.

Side Channel Attacks

PUFs are prone to side channel attacks as well [89], leading to leaking CRP values that enables the attacker to build a model of the PUF. In our proposed method, we suggest one can implement a mechanism that adds correlative noise to the capture power traces through side channel analysis. Here, the designer can employ two PUFs, where one PUF PUFa is the source of key generation and the other one, PUFb, is the source of a dummy key generation. We then propose that the TTP server transfers two C_pack sets, C_packa and C_packb, where R_packa as the offspring of C_packa is logically the opposite of R_packb as the offspring of C_packb. In other words, we expect:

$$r_i \neq r'_i, r_i \in R_pack_a, r'_i \in R_pack_b, i = \{1, 2, ..., (l \times m)\} \tag{4.10}$$

Once C_packa and C_packb sets are received on the device side, they are fed in parallel to PUFa and PUFb, respectively. Once the responses are generated and passed through the majority logic decoder, only the recovered code from PUFa is used for key generation. This mechanism in turn confuses the captured power traces which are recorded during the decoding process, since for each SET operation performed in generating the secret value w, there is a RESET operation performed as well in parallel to generate the dummy value. Such method has already been proposed as a countermeasure for an embedded AES key generation mechanism and assessed against machine learning attacks as well [90].

Replay Attacks

Replay attacks can practically compromise our authentication and key exchange mechanism if there is no watchdog method that assures no challenge value is being used twice [91]. However, we propose using predictive models that are trained on the server side based on the already generated and transferred challenge values. By keeping a record of already used challenge vectors, we can recognize if a generated challenge value is fresh or not. Such solution can be compact and sit aside to the main PUF enrollment solution we proposed earlier.

4.8 Utilization of Simulatable PUFs and its Constituents

Most of our work has been carried out with PUF simulation. We discovered however, that using PUF simulation on an embedded device can be beneficial as well. For instance, early in our work we showed the potential use-case of emulating a neural network based-PUF using some randomly generated values that characterize the random power-up bit values of a weak PUF. Aside to our proposal however, there are some existing works that propose using PUF simulation on board embedded devices such as in.

An attempt we had in this research work was to find a solution for ReRAM PUF which a special kind of memory PUF whose state of each memory cell is programmable based on the distribution of the voltage values read during an evaluation phase [20]. ReRAM PUF normally needs an enrollment phase which is similar to that in an SRAM PUF, where the programmed binary state of the memory cells is captured and later used for encryption key generation. A protocol that uses ReRAM PUF for key generation also is proposed in [92]. Our counter solution for such system was to read the analog voltage values of each memory cell and use them to generate a simulation of strong PUF on-the-fly, and then use the CRPs of the simulated PUF later for authentication and encryption key generation. In this mechanism, we would use the analog voltage values whose distribution over all the memory cells is normal and consider each value a delay parameter in the simulating PUF. Figure 50 shows the schematic of mapping the voltage values into the simulated PUF's delay parameters.

Theoretically this technique seemed potential since we observed the distribution of the voltage values that are normal, and thus it led to yielding good characteristics in terms of randomness and uniqueness which are necessary to a good PUF candidate. However, in practice, such protocol cannot be used for ReRAM PUF due to the fact that the analog voltages are readable only in an evaluation phase where a high current is passed through the component in order to read the distinctive values of the voltage difference between the two ends of a memory cell. Therefore, the simulating PUF can only happen in evaluation phase, and cannot be recreated in a normal operational mode.

Nonetheless, the schematic of simulating PUF using the dispensable analog values from electronic components can remain potential, only if a group of adjusting components dispense values with normal distribution. For instance, we see this a fit in systems where a large group of sensors are used. Having enough of number of sensors to pack

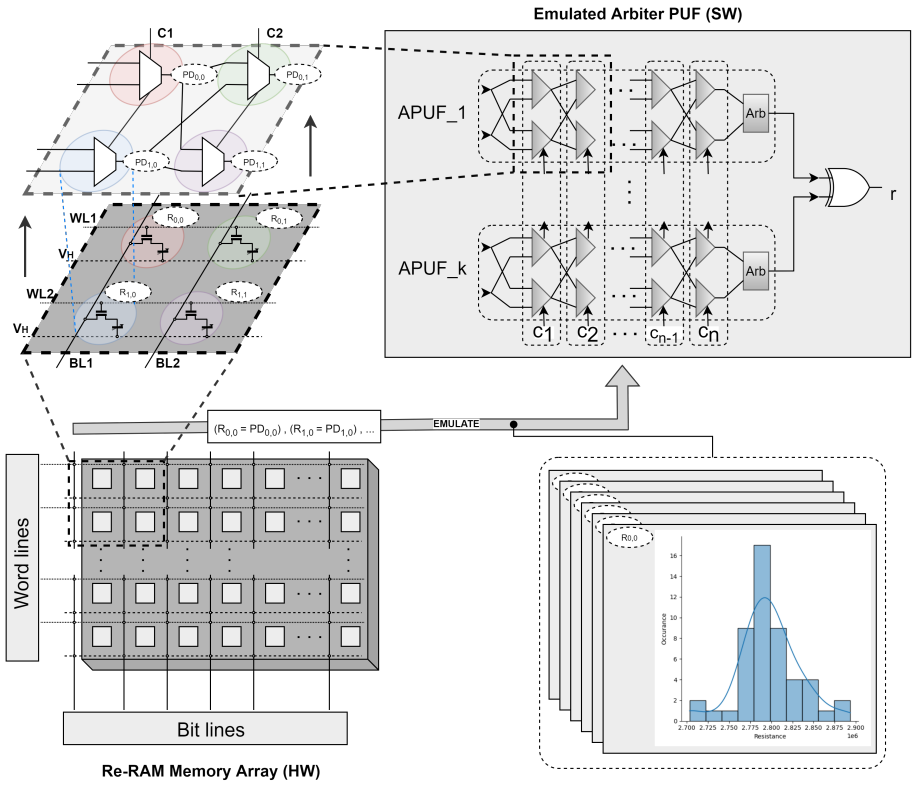


Figure 4.13: Illustration showing a technique in simulating a strong PUF from the analog characteristic of ReRAM memory cells.

into a potential strong PUF (such as a variant of arbiter PUF) can be a good incentive to revise this schematic. We save this however as a later work to be done outside of the scope of this PhD thesis.

4.9 Conclusion

In this chapter we discussed two security protocols based on PUF. Our primary investigation was on Robust Fuzzy Extractor-based (RFE-based) encryption key generation protocol using weak PUF. We followed the concern we observed primarily in the literature on the vulnerability of the RFE method against HDM attacks, and set a goal to mitigate the vulnerability, considering that we could protect the PUF data using PUF data. To do so, we proposed masking the publicly available helper data (the code-offset) using PUF response. In our masking method, we explained that part of the PUF response that is used for key generation can be allocated to generating mask values instead and protect the code-offset which is generated later on from the PUF response and shared

publicly to be re-used during the key-recovery phase. The masking method we proposed as we theoretically showed could increase the entropy of the guessing field in HDM attack considerably. Then we showed practically as well that the success-rate of HDM attack is considerably lowered when we lend more PUF response to masking. The necessity of working on the RFE method in this research work was to get familiarized practically on how one of the existing and pervasively used PUF-based protocols work and how its vulnerability can get in the way of using it nowadays. The masking method we proposed later on came naturally as an idea that can use the existing source of data for key generation and use part of it to protect the key sharing scheme, which was not considered before in the literature.

Then we moved on to set focus again on ML-based PUF computing, the field of operation we already have worked on and improved the throughput. We imagined first of how an ML model of PUF can be used in a security protocol, considering the benefit it brings compared to a database of CRPs. We discussed earlier that the ML model of PUF provides access to the entire CRP space. Therefore, in a way we can consider issuing request for a large number of responses each time we want to open a secure communication session. In other words, we could consider each time a new secret value for key generation, or a fingerprint for authentication. Moreover, the abundance of CRPs could open ways to new secure coding of the secret values and increase their robustness. To realize the full potential, we proposed our version of a secure authentication and key exchange protocol. In the protocol we explained how we realize a key recovery scenario with exchanging only the challenge values. A mechanism which is inherently secure against man in the middle attacks targeting PUF modeling. We also showed a realization of a repetition-like coding mechanism with reliance on a challenge-matching algorithm that dispenses challenges values for exchange whose corresponding response are the same. We evaluated the robustness of the algorithm and showed that we can eliminate the noisiness of the PUF CRP characteristic in mission mode, and the prediction error in the PUF model by increasing the length of the code and be able to keep the success-rate of key recovery at 1.0. We discussed however that the method at this level of implementation may require exchanging a lot of challenges. Later in the upcoming chapter, we discuss what augmentations we can consider in the method to reduce the exchangeable data without compromising the security or the reliability of the method.

The new protocol we proposed in this chapter works for strong PUF variants due to the modelability nature of their CRP characteristic. Such protocol may not be a fit for

weak PUFs, such SRAM PUF. In order to find a mechanism to fit these PUF variants in, we discussed the plausibility of simulating strong PUF over weak PUF data. We briefly discussed it and showed some potential implementations. Such lane of work however needs more investigation which will elaborate on also in the final chapter of the manuscript.

5

Reviewing Challenges and Solutions, and Disscusion on Applications

In this chapter we discuss the next steps that can be envisioned from the endpoint of this PhD study. This includes first to point out the drawbacks and potential vulnerabilities or shortages in any of the methods we proposed, as well as potential occurrences in the future that can comprise our work.

5.1 Introduction

We proposed transfer learning as pre-processing technique in PUF modeling for ML-based authentication. We showed that using transfer learning, we can get closer a solution model for a PUF before training, and thus train the model for a designated PUF faster and with fewer CRPs compared to the conventional model where the initial solution model before training is randomly initialized. In this chapter, we speak more of one of the challenges we can face with Transfer learning in PUF modeling and propose a solution on how to solve it. Then we set focus on sub-space modeling. We provided but a simple version of sub-space modeling. More specifically, we demonstrated the applicability of sub-space modeling on only a variation of Arbiter PUFs, the XOR Arbiter PUF. In this chapter we explain the extendibility of this modeling solution. We also show that applying a modified version of sub-space modeling on subjects with very high CRP complexity and explain that sub-space modeling's expendability is possible and could be required in enrolling strong PUF with highly randomized CRP characteristic.

Moreover, we explain how transfer learning and sub-space modeling can be used together, and where their coupled application is expected the most. We explain that for PUF subjects with very large number of internal modellable components, where modeling in granular subjects requires as well to employ a non-linear model solution, the transfer learning technique can enter and keep the demand for high number of CRPs for training at bay. After addressing the modeling optimization techniques, we proposed in this work, we set to elaborate more on how we can further secure our proposed authentication and key exchange protocol. Despite the claim we have on the security of the exchanging challenge values only, we point out that the raw exposure of these values may become a vulnerability in the future. Thus, we provide a simple coding solution here that can prevent such vulnerability to emerge.

Then we set focus on the application areas of our proposed protocol. Some potential applications that we foresee would benefit from the access to the full CRP space of PUF will be appointed here. At the end of the chapter, we discuss on the future of the FE-based key generation methods, and where our helper data masking which is appointed as a countermeasure for RFE-like key generation methods will end up after a probable expiration of FE-based key generation methods.

5.2 Open Challenges in Transfer Learning

Our proposed Transfer Learning of course has its limits. For instance, we observed that for some cases there exists little leverage that TL can provide. We will point out to that and what can be done to identify such cases. One of the issues we faced with the transfer learning technique in PUF modeling was that in some occasions, the technique was not effective as much as expected in lowering the training dataset size compared to the random initialization. We anticipate that the reason is due the distance between the weight values of the transferred weight package, and that in an already accurate model solution of the target PUF when a random initialization was used in training the model. In other words, the transferred weight data was in the domain of the anticipated solutions that the training model would reach to. In some cases, such scenario did even put the training model in a status far from the anticipated solution, therefore more training data samples were required to lead the training into converging to the optimal state.

To statistically solve this issue, we proposed creating multiple PUF models first us-

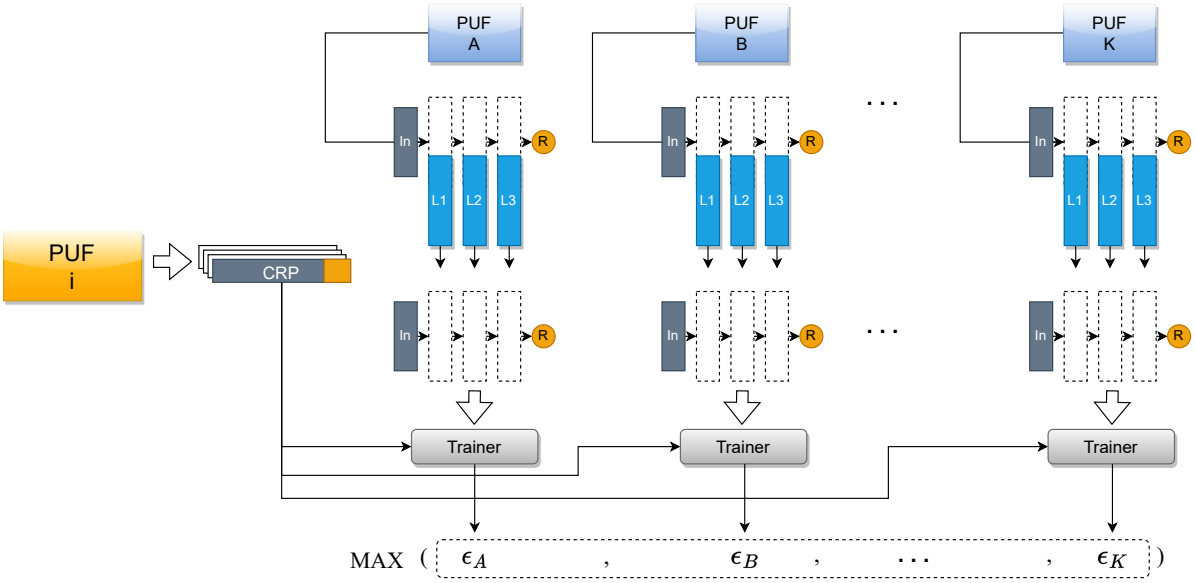


Figure 5.1: Illustration showing the schematic of bagging-enabled transfer learning solution for modeling PUF.

ing the random initialization technique. This would in turn require more data as expected to create the models. However, we take then a copy of the transferrable weight values of all the models and set them as reference for transfer learning. Then in training a larger group of PUFs, if transferring data from one reference did not lead to a convergence, then we attain another reference and transfer its weight values and retry the training. Doing so, we observed that we can indeed decrease the chance of not converging and deliver the target model solution as anticipated with lowered training data. We can see this technique as a bagging solution mixed with transfer learning. Figure 5.1 also shows the schematic of how the bagging solution works with transfer learning in modeling PUF.

A more sophisticated take on the bagging solution is to consider each successfully modeled instance a reference, and so increase the options for referencing for future modeling targets. Nonetheless, as the options increase, it may be possible that the time of training increase as well for the exceptional cases which fail to converge into an optimal state during training. Then a debate may arise, on whether its essential to keep performing the bagging solution, or normally capture more data and train the model.

We anticipate also that there may be non-statistical solutions with transfer learning in PUF modeling which enable the solution as a more deterministic pre-processing must-do when PUF enrollment is done using machine learning. The potential of trans-

fer learning nonetheless exists with such use-case scenario (PUF enrollment with ML) since many of the enrolled PUFs share high level features that as discussed can aid the training if these features are transferred from already optimally trained models, to new models ready for training.

5.3 Open Challenges in Subspace Modeling

Sub-Space modeling hasn't been investigated deeply in this work. While with the primary investigation, we observed promising results, we anticipated some modeling mechanisms that are driven from sub-space modeling and can potentially lead to better results. We discuss that here. Sub-space modeling can also be a costly solution in terms of real-estate and the extra effort that is required to enable extracting internal CRP values. We discuss that here with some possible improvements that can happen.

We observed previously that in modeling XOR Arbiter PUF with large number of XORs, the model accuracy cap is lowered when we consider modeling each arbiter PUF separately. We could estimate that we will get about 1% accuracy per Arbiter PUF chain. Therefore, for a 10-XOR Arbiter PUF, we observed a maximum 90% prediction accuracy. Now if we target 90% accuracy as the minimum acceptable accuracy, then sub-space modeling at its current state cannot handle modeling XOR Arbiter PUF with more than 10 Arbiter chains. Same is true for any other PUF structure that represent a similar structural complexity.

To mitigate this issue, we anticipate that if we lower the divisor, we can maximize the prediction accuracy cap. For instance, for a 10-XOR Arbiter PUF, instead of dividing the target PUF into 10 sub-components, we divide by 5, thus we get five 2-XOR PUFs to model.

Another anticipated challenge with sub-space modeling is how we can divide more intertwined structures into smaller sub-components? For instance, how we can apply sub-space modeling to a feedforward Arbiter PUF? Or what happens when we have a target Arbiter PUF complex that includes feedforward sub-parts and XORing sub-parts as well? Addressing these questions are more of design solutions. For instance, for feedforward Arbiter PUF, we can consider the schematic in Figure 5.2 (a). And as well for the complex Arbiter PUF mixing feedforward and XOR compositions, we can follow the schematic in Figure 5.2 (b).

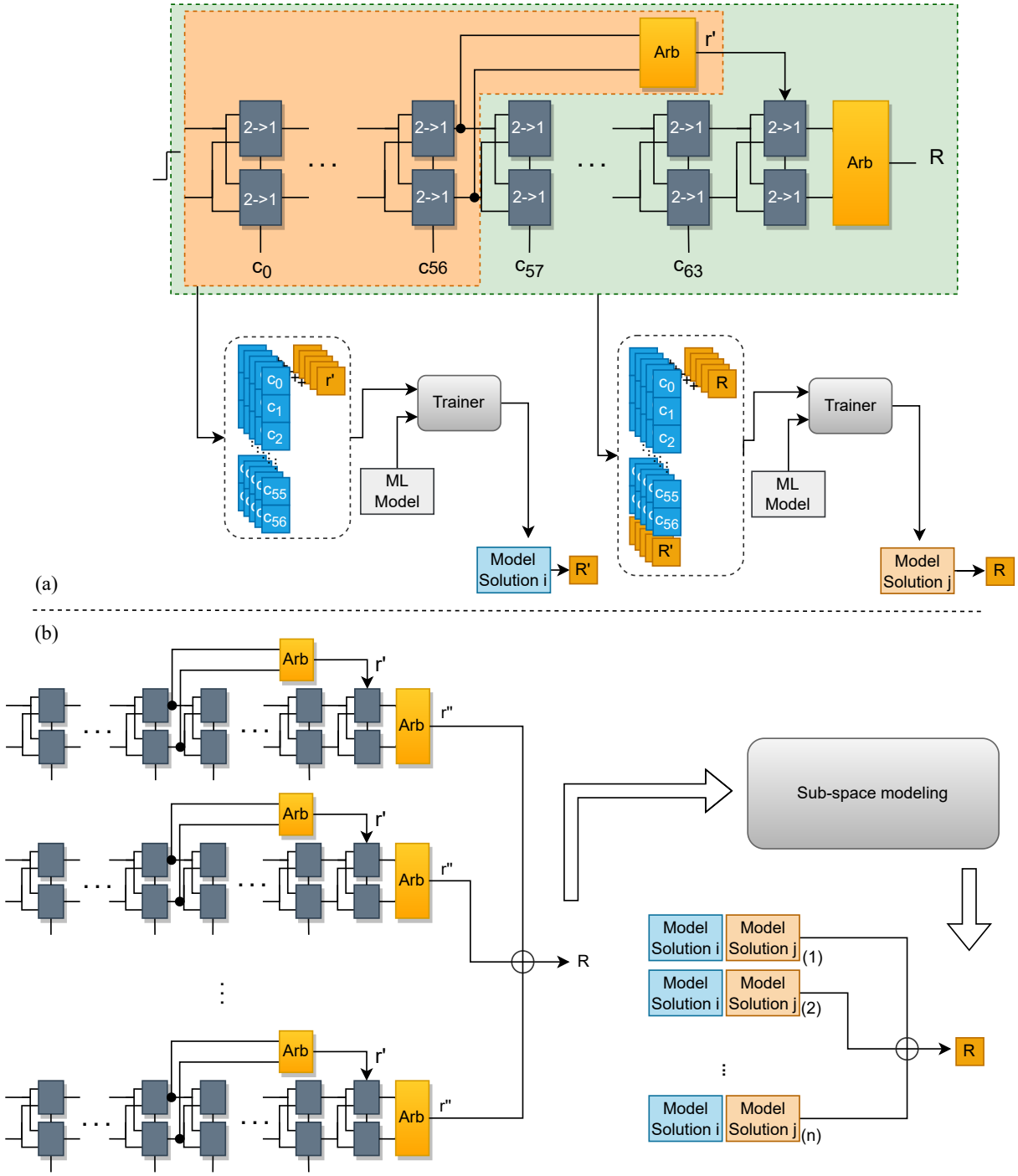


Figure 5.2: Illustration showing (a) the design solutions for applying sub-space modeling for feedforward arbiter PUF and (b) complex arbiter PUF mixing feedforward and XOR compositions.

Table 5.1: Various modeling compositions using sub-space modeling and transfer learning to model a 40-XOR Arbiter PUF.

Suggested Composition	**Heuristical estimation of ϵ**
40 × LR model of Arbiter PUFs	$\approx$ 60%
20 × MLP model of 2-XOR Arbiter PUFs	$\approx$ 80%
10 × MLP model of 4-XOR Arbiter PUFs using TL	$\approx$ 90%
8 × MLP model of 5-XOR Arbiter PUFs using TL	$\approx$ 88%

5.4 Mixing Modeling Solutions to Yield more Optimal Results

We introduced Transfer Learning and sub-space modeling so far as non-overlapping solutions both targeting to lower the training data and to yield the solution with desired accuracy as fast as possible. In this work, we did not mix these two techniques together. But here we can discuss that such approach can be possible and can lead to improved outcome.

For instance, in the case of very large XOR Arbiter PUFs, we suggested using sub-space modeling with smaller divisor to avoid drastic decrease in the maximum achievable prediction accuracy. We can anticipate that at some point, if the design complexity of the target PUF is very large, the smaller components may in turn be complex enough for individual training. In such case, we can employ transfer learning to model the smaller sub-components. Let's for instance consider a [N * 10]-XOR arbiter PUF for N > 1. This is a very large design for a singular strong PUF. We can anticipate that the randomness of the PUF very large and thus makes the design very interesting in case the security of the overlying system is at stage. To model this for enrollment however, we would need to carefully apply the sub-space modeling and transfer learning to yield a model with highest accuracy. To tackle this, we investigate multiple possibilities in modeling a 40-XOR arbiter PUF. Table 5.1 shows these possibilities.

We came so far with these improvements in modeling PUF and talked about specifications that can be considered only in case of modeling PUF for enrollment. Of course, our suggestions for improving the modeling of PUF here are not ultimate solutions. We can anticipate that more complications would exist targeting novel PUF architectures. Our purpose of showing the possibility of different takes in modeling PUF here was to show that potential techniques exist which can make the enrollment of PUF with machine learning easier. And solutions as such are not crafted only for specific types of

PUFs such the Arbiter PUF family we worked on dominantly in this work. We encourage that strong PUF enrollment using ML need to be investigated on other variations as well and identify the complications and provide solutions to mitigate them, similar to what we did with sub-space modeling and transfer learning.

5.5 Anticipated Improvements in the Secure key Exchange Protocol

Our secure key exchange protocol can also benefit from some improvements. For instance, we know that the primary form of our protocol regularly needs many challenge exchanges to assure the same secret value is recovered on the PUF-enabled device. This makes the method running slower in certain conditions. We discuss that here and open the field for potential solutions that can emerge to mitigate this issue.

In our ML based key exchange protocol, two key factors affect the probability of successfully recovering the secret value: 1) The stability of PUF response 2) The accuracy of the PUF model response prediction. Expectedly, when the probability of response flip in PUF increases (due to environmental variations or other physical perturbation), correspondingly the success-rate of key recovery can decrease, if the capacity of the error correction code is fixed. That is why we proposed first to increase the repetition length in case of observing the success-rate of the key recovery is falling below 1.0. Nonetheless, a concern raises in the case of mandating to exchange large number of challenges, that the size of exchanging data for one key recovery may not be justifiable. Moreover, the exposure of too many of the raw value of the challenges may raise some security issues. While still the response value is not part of the exchanging data, for those of attack models that have physical access to the PUF, the only target would be to audit the response values somehow, since they already can extract the challenge values from the communication channel. One of the late improvements we thought of adding the primary design of our authentication and key exchange protocol to face the mentioned concerns, is the following:

Instead of exchanging raw challenge values, we consider sending a sorted 2D list of a range number of byte-size digits alongside with a reference challenge value. The 2D list can be represented as in the Figure 5.3. Here, each digit represents the index of a challenge with respect to the reference challenge value. Each row of the list represents

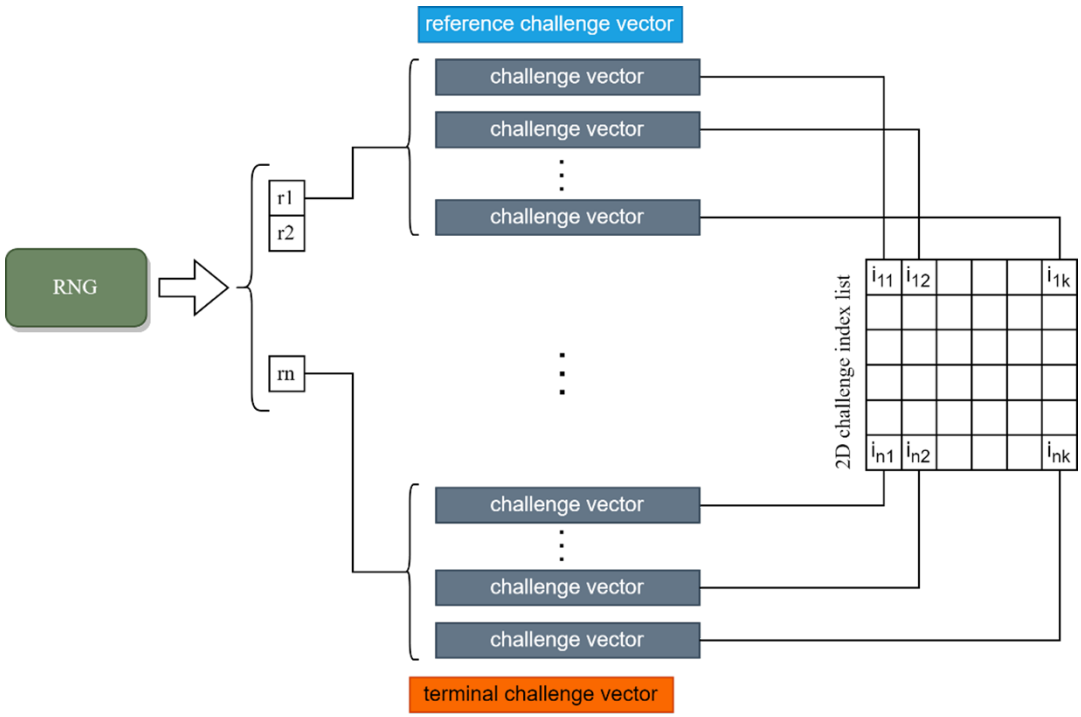


Figure 5.3: Illustration showing the organization of the 2D challenge index list.

the binary value (0,1) of the secret key at the corresponding index, and each column represents a challenge for that class of binary value, whose response according to the sourcing PUF is the same binary value as in the corresponding class.

The 2D challenge index list should in turn be considerably lighter than the primary solution we had to solely exchange the challenge value. For instance, for a 128-bit Arbiter PUF variant, if we consider creating a secret value of size 16 bits, with repetition code length 10, if we use the primary exchange method, we would need to exchange 2.5 KB of data comprising challenge values, while using the 2D challenge index list, the exchanging data size is reduced to 0.15 KB.

5.6 More Opportunities with Coupling ML & PUF

So far we talked about the benefits of using an ML model of a PUF. We explained that initially the model of the PUF will replace the CRP database, with the aim to provide full access to the CRP space of a PUF, given that the model as a solution is compact and storable (which normally is, given the size of the models created of PUF so far). However, it is notable that our discussion on such potential pivoted around strong PUF,

which is the variation of PUF that usually favors/suffers the modelability using ML. This on the other hand is not applicable on weak PUFs. We mentioned it briefly in the previous sub-section. But to elaborate more on it, one can imagine that the only CRP characteristic a weak PUF has, is the relationship of the memory address, and the extracted vector of bits from the memory, which are the power-up values of the memory cells designated to that memory address. We can assume that the memory address is the challenge, and the binary vector is the response. However, we cannot assume that an ML model can provide a solution in which the relationship between the memory addresses and the responses seemingly have hidden learnable patterns. Therefore, even the effort to build such models will fail.

Nonetheless, there can be another take in using ML and the weak PUF data. Recently a work done by Karimian et al was published in 2020, proposed using deep learning to create an authentication solution. In their work, they employed a CNN model which analyzes the PUF response (the binary power-up values of the memory cells) in form of a 2D image and associates the learned features to the ID of the owning PUF. In such solution, the model will be able to identify the PUF based on the PUF response it sees. This is a potential solution which does not model the PUF but builds a predictive model that can identify PUF. However, the solution in turn lacks security due to the fact that the PUF-enabled devices need to send their PUF image entirely to the server for identification, which in turn exposes the image to the public and it can be exploited by replay attacks. At first sight, designers can suggest encoding the image first and then sending it to the server and decoding it there before identification. However, as we already know, the encoding itself requires a source for the key value to cipher the images, and that as a requirement can add complexity to the solution.

To deal with this, we anticipated some adaptive changes to the way the identification can be taken cared of. For instance, we proposed sending a fraction of the image to the server in [93], while the server has a reference image of the PUF at sight, and the fraction can replace its corresponding part on a copy of the reference image on server, and the product then will be sent for identification. Such method in turn will not expose the entire image to the public, and with the fraction only exposed, attackers will have less of success chance to re-use the value for a replay attack. To assure this for instance, we proposed using a different fraction of the image at a time. Figure 5.4 shows the general schematic of the method we proposed.

Another interesting idea that we could briefly develop was to use the same tech-

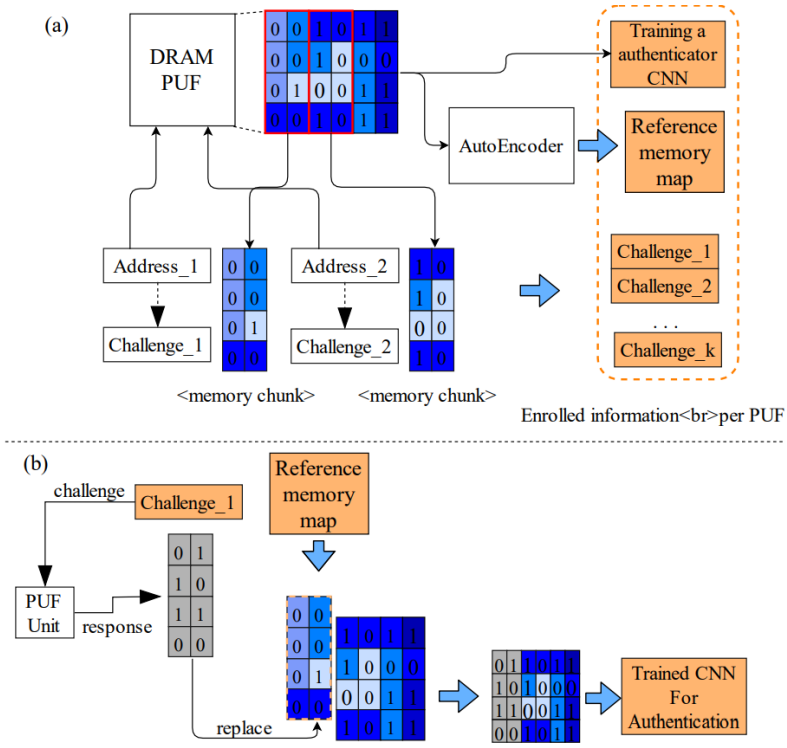


Figure 5.4: Illustration showing our improved way of PUF authentication using CNN.

nique of PUF authentication for strong PUFs. We proposed it in in [93]. To do this, we imagined creating PUF images of using multiple responses captured from a range of consecutive challenges and attempt in associating the image to a PUF identity. Since the expectation on the PUF image even for the memory PUF is that the elements are randomly set and have no correlation to each other, we can expect the same if we make PUF image by using multiple binary responses of a strong PUF. Since for a good strong as well we expect the same randomness. Figure 5.5 shows the schematic of our proposed DL based authentication method for strong PUFs.

The only matter in this method is since strong PUF can generate very large number of CRPs, then how many images we can associate to a single PUF identity ? Can a single CNN model handle multiple images for a single PUF id ? or should we need multiple models ?

Another exotic method we ventured on was to create a new PUF using the power-up values of the memory cells [94]. We saw this as a potential to create a design that can benefit from the random memory cell values and represent a strong PUF, meaning that it can generate a large number of CRPs. In such case, the PUF on the memory is the

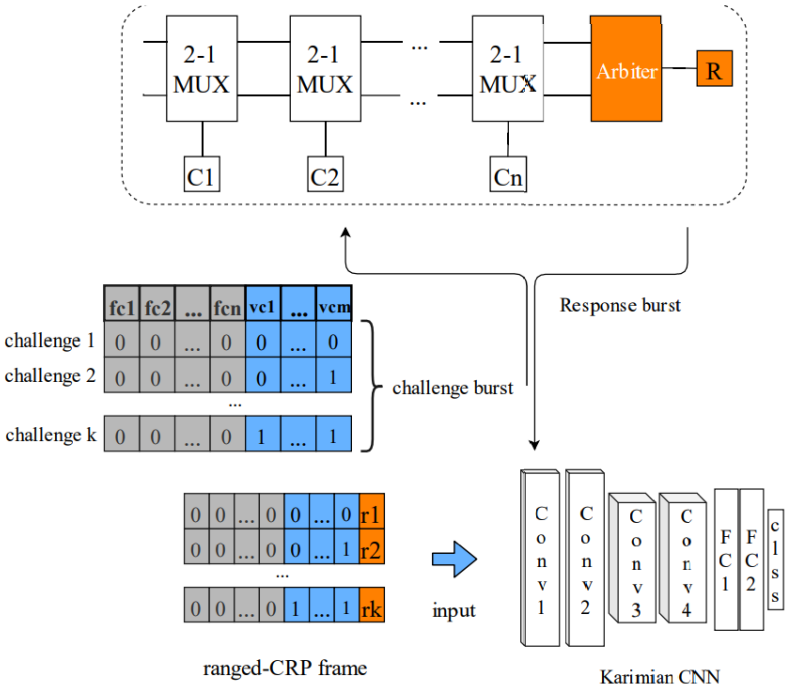


Figure 5.5: Illustration showing our proposed method to use Karimian's CNN model for PUF authentication for strong PUF.

power-up values of the memory cells, but the design on top which utilizes the power-up values. To do so, we proposed to employ the structure of an ANN, whose synaptic weight values are the power-up values of the memory cells. Theoretically, it seems a doable task to create a PUF out of a neural network structure (similar to an XOR arbiter PUF) and it has been explored once in [95]. However, our primary assessment showed that the characteristic is not ideal enough (based on the PUF metrics we introduced in the previous chapter) to candidate a PUF for identification purposes, but good enough as a random number generator (a side application of PUF in general that is discussed in the literature as well). Developing such methods up to a point of applicability was not the focus of our work. Recalling that one of the main focuses we have in this work is the CRP dataset management which mainly concerns the strong PUFs. Therefore, we left these ideas remain at the state-of-the-art level of implementation.

5.7 Range of Applications using PUF

PUF is considered a cyber-security primitive. So, in the nature of that, it can be used within any application that is pivoted on cyber-physical systems. Since PUF has been

around for more than two decades, and inclusive and deepened studies on many aspects of the PUF has been done since then, it is natural to expect that PUF is useful and is actually used for a wide spectrum of applications. Given the type of the PUF, and the size of the CRP space for each PUF family and its sub-variants, specific applications can be designated to them that utilize their maximum potential. Here we will point out to some of the prevalent applications of PUF and its sub-variants.

IoT

Pervasive application of PUF can be found in IoT systems where employing a security protocol that is efficiently defined for resource constrained devices is of the key requirements. Applicability in IoT however has a large spectrum of field specific use-cases.

In the large variations of IoT edge devices, there exist a sub-class of devices which have small form factors and are commonly battery-depended on to operate. This means that they have to manage their power consumption strictly, and so inherently, most of these devices come with hardware components which are operational in low power mode. The silicon real estate in these devices is limited also, thus there would be less room for components that are either not efficient for small form factors, or not well fitted in low power operational mode. Mostly these devices are packed with sensory components and communicational components to send the sensory data to a centralized computer which runs the necessary algorithms. Since communication is a key part in IoT devices, it is also important to take care of the data transmission security, and the security of the entire ecosystem in general. Given the physical restrictions mentioned in prior for the IoT edge devices, system designers are more leaned towards employing security primitives that have low hardware overhead while containing the security of the system. PUF is such component for designers to employ in IoT systems. Not only PUF can fit well within the physical restrictions, but also due to its tamper resistant nature, it provides security in the physical level as well as in the communication level between the IoT devices. This means that with PUF employed, it would be harder for attackers to physically clone the PUF-enabled devices. There are many discussions and proposals of security protocol for IoT using PUF, of which some are mentionable here, such as [96] which proposes an authentication protocol for IoT using PUF. This work is a secure version of an earlier proposed authentication method for IoT using PUF which was proposed in [97].

Industry 4.0

Industry 4.0 refers to the fourth industrial revolution, where the main changes to the industry are larger inter-connectivity and smart automation. In this concept, cyber-physical systems and IoT play a major role. Smart factories, at the center of Industry 4.0, comprise many interconnected devices that automate the manufacturing process alongside a smarter environmental monitoring and self-monitoring scheme which no longer necessitates the presence of human intervention. Nonetheless, such concept is still in the maturation phase and in that it still requires more optimal security protocols that fit the needs of the industry such as lower power consumption and hardware overhead. Such need has been instated in many recently published papers such as [98] where the authors elevate the importance of PUF-based protocols for Industrial Internet of Things (IoT) and the industry 4.0 in general.

Smart Grid

As another domain specific implementation of IoT, smart grid as well requires optimal solutions for security. To synchronize power distribution between consumers from the power source, power grids require Smart meter (SM) devices. SMs are fundamental networking devices which are linked to the power grid through networked communication. SMs though are devices that are susceptible to physical attacks and attacks that target the communication between them and the central control system near the power grid. To ensure the authenticity of the exchanged metric data, advance metering infrastructure (AMI) is employed which in itself employs authentication and key agreement (AKA) methods to provide security [99]. PUF here comes in as a suitable security primitive due to its inherent resilience against physical attacks and up to now several works proposed PUF-based authentication and key exchange protocols for smart grids [100], [101].

IC Traceability and Blockchain

One of the main concerns in IC manufacturing process is the risk of tampering with the IC or counterfeiting the silicon chip housed products. Actors who perform device tampering or counterfeiting can have several interests, such as cloning the device, or injecting their malicious programs. Thus, it is of outmost important to implement robust traceability techniques to secure the supply chain and avail the tech-owners to

ensure counterfeit avoidance and detection. This should in turn benefit not only the supplier, but the costumer as well. Traceability in IC manufacturing refers to the combination of the ability to know the current possession of a product at all times (track) and the ability to find the origin, ownership history, time spent at each point (trace), by means of recorded identifications. Such concept can be mapped on a blockchain-based ownership management system, where an immutable database (the blockchain) that maintains a continuously growing list of transaction records secured from tampering and revision is provided on the supplier's IC tracer system. To satisfy this system with a secure implementation, PUF can be employed as the root of trust. Due to its low hardware overhead, low power consumption and tamper resistant nature, PUF can be seen as an ideal choice for root of trust for a wide variety of silicon chips, given that such implementation of root of trust for IC traceability has already been widely discussed in several works in the literature such as in [102].

Healthcare

E-health nowadays is a trending subject both in the industry and academia. Smart healthcare is inevitably borrowing a lot from electronic devices and computing systems for use-cases such as patient vitals monitoring and motion tracking, to electronic implants that replace damaged internal organs and prosthetic limbs that replace amputated parts of the body. Such close-up contact between biological organs and electronic devices demands certain limitations to the design and operation of these devices. It is expected naturally that these devices are operated in low power mode and are also fitted well on, or inside the body. As this field is directed towards maturation, many studies have proposed so far different form factors and operational modes for such devices that follow the physical standards for E-health. Nonetheless, these devices have the ability to connect to a central station to which they transmit vital or physical activity records. Such nature in itself requires security as well due to the fact that such information is by nature personal and thus should not be exposed to the unauthorized public. For such requirements, PUF again can be found as a useful security primitive and indeed several works have already investigated possible AKA protocols that can use PUF, such as the work done in [103].

Agriculture

Agriculture is another domain specific use-case of IoT, yet one of the most important ones. The application of IoT in agriculture helps with monitoring several production field parameters such as soil moisture, temperature, pH, humidity, and gas, aside to the automation of cultivation, field maintenance, and planting. Analyzing these parameters is an important procedure to help maintaining the ideal conditions for maximum crops productivity. Monitoring such parameters needs sensory devices that are operable in the field for a long period. Such devices have low computation power, and mainly act as data receivers and transmitters. Therefore, they need security protocols implemented which robustly and resource efficiently ensure that the transmitted data are not tampered with or altered for malicious use-cases. In fact, the U.S. Department of Homeland and Security report on security issues in Agriculture highlights the importance of data security, confidentiality, and integrity in agriculture [104]. This means that serious attention to the security in the field of smart agriculture is needed. PUF in such scenario can come of good use, as is explained in a recently published work [104], where the authors propose an authentication method based on PUF for smart farms.

Self-Driving Vehicles and smart transportation

Smart transportation is another important field of application for IoT. Trending subjects such as vehicular network, traffic monitoring, and autonomous charging are of few subjects that benefit from IoT in the field of smart transportation. Unlike traditional transportation, smart transportation requires a more secure infrastructure for data exchange and rapid validation to ensure data validity. Since all entities such as vehicles, traffic monitoring facilities, charging stands, etc., are interconnected and constantly in communication, they represent themselves as interesting points for breaching for potential hackers, and one can assume that breaching into such network of things can yield catastrophic outcomes such as disturbing the traffic in urban areas or autobahns, or misrepresenting key information in Vehicle-to-Vehicle connections that may lead to accidents. Such eco-systems thus can get a good use of PUF to ensure a robust security primitive is in place for communication. For instance, Sadhu et al. proposed a novel PUF-based vehicle authentication method using Ring Oscillator PUF as discussed in [105]. This method in turn is useful for rapid authentication while also preserving the secureness of the communicated data and assuring the authenticity of the communicating device

as the vehicle. Another work has been conducted by Pudi et al. [106], where he presents an authentication protocol using RO-PUF that augments the security and authenticity of video and imagery data that is transmitted between the Vehicle, the automated traffic monitoring system, and the base-station. Aside to authentication, PUF shows potential in generating pseudonyms as discussed in [107] by Petit et al., which is useful for secure communication between vehicles without disclosing any private information of each vehicle.

Smart Home

Another interesting application of IoT is in Smart Homes. IoT based Eco-systems in residential places can provide remote access to smart facilities inside the residential for the residents who may at times reside in any place outside of the residential. Such convenience of course needs to cooperate with secure mechanisms that ensure the authenticity of the accessors outside of the residential, while also ensure that no unauthorized device is trying to impersonate itself as part of the residential's facilities. Noting that in such eco-system, we are dealing with variety of devices from small sensors to video-based monitoring facilities, to mechanical devices such as smart door locks. Many of these facilities may be battery operated and provide small real estate for computational components. Such limitations again could candidate PUF as an ideal security primitive to take place. As an instance of that is a recently published paper by Xia et al., who proposes residential device authentication based on PUF [108].

Smart Card

Smart Cards nowadays are inevitable parts of our lives providing us with the ease of access to our financial services. Due to the nature of the smart cards being tangled with the financial properties of their users, they need to be protected as efficiently and as robustly as possible. It is apparent also that smart cards cannot hold very complicated cryptographic computational blocks due to their small form factor. Thus, the technology needs to upgrade to show resilience to new potential attacks. Conventional security concepts up to now are based on Elliptic curve cryptography, which are vulnerable to the card lose attack and desynchronization attack, where some schemes add a random number in verifier-value to resist the card lose attack and store both the old and new pseudo-identities between authenticator and the corresponding authenticated party to

withstand desynchronization attack. However, the random number stored in the card memory can be extracted and the new conversation may be blindly blocked by adversary [109]. PUF can be of good use here to provide a resource efficient security primitive. An example of that is proposed by Chen et al. in [110] where they propose a novel authentication protocol that utilizes PUF and elliptic curve cryptography (ECC) to protect the random number that is generated within the card.

With the given examples in IoT based PUF applications, it is evident that PUF phenomenon is indeed a potential security primitive due to its physical and computational capacities. Yet despite the prevalent use-cases of PUF, the protocols that are based on PUF are still on their way to maturation. Many studies have been conducted that show the vulnerability of PUF-based protocols and the complementary on the other hand present novel techniques to mitigate these vulnerabilities. To take a deeper look into how security protocols use PUF, we explain in the following the fundamentals of PUF-based protocols and the variations of them.

5.8 The Future of the RFE-like protocl and our Helper Data Masking Countermeasure

Our helper data masking method also have a downside. A bold challenge in it is that FE methods in general are going to expire soon due to the public exposure of the helper data. This means that our method may not be useful anymore for key exchange protocols. This is a global issue that all FE-based key generation methods are facing. We discuss that here and try to depict the expected horizon for our masking-based countermeasure for other PUF-based algorithms.

The main security problem in Fuzzy Extractor methods is the exposure of the helper data. We have seen that code-offset is an important element in recovering the secret value that is used in creating an encryption key. We also saw that HDM Attacks can embed an error vector to into the code-offset which later during the decoding procedure lowers the guessing entropy on the attacker's side. This in turn enables the attacker to manage creating a mutual encryption key with the PUF enabled device with fewer trial and errors and impersonate itself as the server and exchange confidential data.

There are also other types of HDM attacks that exist as we discussed in Chapter 3.

In all of them, mutually the problem is that the helper data in turn presents itself as a doorway for attackers to breach into the key recovery procedure. We foresee that future FE methods are forced to no longer incorporate public exposure of core helper data such as the code-offset. Most probably, this will lead to either expiring the FE-based key generation or change the vitality of the publicly shared helper data. For instance, one can imagine that helper data represents some meta information about the generatable key which is not directly affecting the key recovery outcome.

Our proposed helper data masking countermeasure on the FE-based key generation methods using PUF may be considered an incremental solution. This would mostly be due to the fact that the overall interest in using FE-based methods for generating key from biometric data is decreasing [ ]. Nonetheless, we showed that for the proportion of existing key generation protocols using RFE-like method coupled with vulnerable coding algorithms, our solution can be considered as a low-cost adaptation to reduce the success-rate of HDM attacks. It has been already discussed in [ ] that despite the vulnerability against HDM attacks, RFE-like methods with primitive coding algorithms are interesting for their low-cost implementation. Such interest may continue to exist for IoT edge device security. Therefore, in that lane of application, we can consider our countermeasure to exist as well since the cost of employing it is only a modification in the implementation of the key generation code, where instead of using the entire extracted secret value from PUF for key generation, we preserve a portion of it for masking, and we follow the same procedure of extracting the code-offset for the mask values as we do for the secret key.

In general, we can also anticipate that masking solution could be around for a longer period due to their effectiveness against machine learning attacks. Further obfuscation of the exposed data in turn is a key countermove against the power of prediction in the ML models [ ] as it can potentially lead to needing more training data until convergence, or completely randomize the data so that no ML model with high prediction accuracy can be obtained. Moreover, there are several masking solutions proposed in PUF-based based key generation algorithms such as in [ ] and [ ]. We can anticipate that a more efficient way of implementing mask values for algorithms as such can be the way we proposed in extracting and recovering the mask values from a PUF source. In such scheme, we can still be coherent with the recent of design and do not use an external storage to keep the mask values in.

5.9 Conclusion

This chapter mainly focused on some of the late blooming ideas that can augment the initial proposals that was given in the early stages of this work. Elaborating on how we can further sophisticate transfer learning optimization, and sub-space modeling, and how we would be able to use them conjointly in enrolling very large strong PUFs, were parts of the discussion in this work. We also proposed a structural optimization for our proposed authentication and key exchange protocol and showed that we can even hide the challenge values on plain sight during the communication for authentication and key exchange. While theoretically shown applicable, we encouraged that further evaluation of the secureness of the proposed modification is required. Al though the proposals were shown firmly here, the mere intention of this chapter was to show that the proposed methods here are pioneering and can be the base-ground for further augmentations. At the end we pointed out to some of the potential use-cases of our proposed protocol, with this point in mind that all the use-cases in their core will benefit from the full access to the PUF CRP space. Finally, we explained that FE-based methods may be at the end of their use-fullness due to the increasing stress on their vulnerability. However, for the sake of their simplicity in design and operation, they may keep existing in resource constrained IoT systems, and for that, our countermeasure as well can stay with to provide an incremental level of security. Further on we mentioned also that other PUF-based security algorithms exist that promote the usage of masking as a countermeasure, and so we pointed out that our masking mechanism can be considered within these algorithms as well since in our mechanism we assure the coherency of using PUF as the primitive.

Final Conclusion

The research work developed at the core of this dissertation is two folded. The primary investigation was set on the viability of using PUF modeling with machine learning for enrollment. For this we first discussed that we need a methodology to include ML-based modeling of PUF into the enrollment procedure. We explained that ML-based modeling is a potential method to model the PUF, however it is empirical, therefore we need a methodology that considers the possibility of trial and error in providing a model of PUF using ML, but maximizes the probability. To do so, we justified that we need an evaluation phase wherein the PUF subject is analyzed either with set of simulated or real-world data to unravel the optimal values for hyper-parameters in modeling, and control parameters in the enrollment procedure. We explained also that the optimality of these values depends on the desire of the user, either targeting to obtain the maximum accuracy for the predicative models of PUF, target the combination for parameters which leads to providing models of PUF faster, or the maximum size of CRP dataset size that yields the minimum acceptable accuracy. We also explained that other scenarios can exist which exist in between the spectrum of possible outcomes.
After introducing the methodology, we moved onto explaining two optimization techniques in modeling PUF with increased complexity. The necessity in optimizing modeling PUF with increasing complexity is not only assuring the security of the underlying system, but also assures the facilitation of the enrollment of the PUF. To optimize PUF modeling for enrollment, we introduced first the concept of transfer learning. We explained that transfer learning realizes trained data reusability as a mean to reduce training data and training time. We then proposed using transfer learning with multilayer perceptron (MLP) model, which is an artificial neural network structure. In specific, we proposed practicing transfer learning with a recently proposed MLP model which is adaptable to the structure of the target PUF. We showed that together, the adaptive MLP model and transfer learning are more potentialized in modeling PUF with reduced training data. Such potential enables modeling PUF with increased complexity with knowledge that is only known by the manufacturer or the designer of the PUF. Moving forward, we also proposed sub-space modeling, a divide-and-conquer

oriented technique which proposes peripheral structural modification as a mean to provide internal PUF response coming from sub-parts of the PUF. Such schematic allows modeling each sub-part separately, which in turn leads to requiring lesser training data to model the overall PUF target. We also ascertained empirically that sub-space modeling can indeed decrease the training dataset size compared to targeting the entire PUF CRP characteristic at once. Together, the two optimization methods, transfer learning and sub-space modeling, are potential co-ops that can arise certainty of modeling PUFs with very large structural complexity with high prediction accuracy. We elaborated on this speculation in the last chapter.

The second fold of the research work is set on developing secure and resource efficient protocols. First, we investigated one of the conventional key generation methods for PUF-enabled systems. Fuzzy Extractor is the method which is promoted in both the industry and academia to be used for encryption key generation on noisy sources of secret primitives such as PUF. We then explained the vulnerability robust fuzzy extractor-like (RFE-like) method to HDM attacks and explained that the vulnerability of the publicly available helper data as part of the method schematic needs extra level of protection against HDM attacks. To which end then we elaborated on our proposed adaptive helper data masking countermeasure, which is a masking mechanism directly targeting to independently increase the entropy of the guess field on the attacker's side. We then empirically showed that our method can indeed increase the entropy of the guess field of the HDM attack, which in turn drastically decreases the chance of success in the attack. The attempt to create this countermeasure was the first step to get involved with developing a resource efficient protocol based on PUF. Although the final outcome was a countermeasure and not a protocol, the effort taught us that it is possible to count on PUF to be the source of masking which in itself is a promising act which can be potential in not just FE-based methods, but other PUF-based protocols which need further protection at data level.

At the second step in protocol development, we set focus on a blueprint of an entire protocol from scratch which now is based on the coupling of PUF and its equivalent ML model, a rare move yet potentialized, which we have made prior preparations in terms of enrolling PUF with ML. We explained that the existence of an equivalent model of PUF on a TTP server enables creating locally correctable codes, in sense that the code itself does not need to be exchanged through channel, rather a set of challenges that lead to the code via PUF and the equivalent model of PUF, are exchanged. We explained first

a challenge matching scheme we enable such exchange of codeword which in turn is secure against model-building attack since no response with the challenges exchanged are coupled. Then we explained how we can harness a repetition coding into this exchange method which in turn enables us to cultivate robust secret values usable for encryption key generation. We then demonstrated the entire procedure of the exchange and the coding and decoding mechanism, and later an authentication and key exchange procedure that utilize the recoverable secret key. We also empirically showed that the method can in turn generate and exchange 100% reliable secret values, of course by counting on the trade-off between the reliability and length of the repetition code. This would mean that for conditions in which we speculate that the reliability of the recovering secret value is subject to deterioration, we increase the length o the repetition code to amend for the potential loss of the reliability.

We also explored on some of the potential yet side-missioned ideas of coupling PUF and ML. For instance, at the primary phases of the work, we investigated we can use deep learning for strong PUF authentication, a method that has already been proposed for weak PUFs by Karimian et al. We also proposed that we can create strong PUF in the structure of a binary classifier neural network using the primary random values of a weak PUF source, a mechanism which in turn enables cultivating more from a weak PUF sources than just a single read-out. For the binary classifier however, we explained that the initial idea is good to be investigated to develop a random number generator, and not yet for security protocol.

We also set a brief focus on how we can link the contributions of PUF modeling which we dominantly explained for strong PUF families due to their very large CRP space, to the Weak PUFs as well. To do so, we explained the possibility of simulating or emulating strong PUF over weak PUF data, an idea which we introduced and explained for potential device types such as Re-RAM, but due to impracticality of using the raw values of the device in normal operational mode, we were unable to further continue in our investigation. Yet we explained that such idea can be explored further using other device types such as a local ecosystem of sensory components which can represent analog values. In general, this work aimed to investigate the potentiality of coupling PUF and ML. A potentiality which has been partially harnessed in the optimization methods and protocols we proposed and explained here. Some of which we elaborated on experimentally and empirically, and some we presented as potential ideas. Later works that are set to run from this research work on will have at hand a solidified and opti-

mized methodology for strong PUF enrollment using ML, and a baseline protocol for key exchange and authentication.

BIBLIOGRAPHY

[1] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: a tutorial", *Proceedings of the IEEE*, vol. 102, *8*, pp. 1126–1141, Aug. 2014, ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2014.2320516. [Online]. Available: http://ieeexplore.ieee.org/document/6823677/ (visited on 03/07/2021).

[2] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A puf taxonomy", *Applied Physics Reviews*, vol. 6, *1*, p. 011 303, 2019.

[3] A. Shamsoshoara, A. Korenda, F. Afghah, and S. Zeadally, "A survey on physical unclonable function (puf)-based security solutions for internet of things", *Computer Networks*, vol. 183, p. 107 593, 2020.

[4] S. Khalfaoui, J. Leneutre, A. Villard, I. Gazeau, J. Ma, and P. Urien, "Security analysis of machine learning-based puf enrollment protocols: a review", *Sensors*, vol. 21, *24*, p. 8415, 2021.

[5] J.-Q. Huang, M. Zhu, B. Liu, and W. Ge, "Deep learning modeling attack analysis for multiple fpga-based apuf protection structures", *in 2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology* (*ICSICT*), Qingdao: IEEE, Oct. 2018, pp. 1–3, ISBN: 978-1-5386-4440-9 978-1-5386-4441-6. DOI: 10.1109/ICSICT.2018.8565728. [Online]. Available: https://ieeexplore.ieee.org/document/8565728/ (visited on 03/07/2021).

[6] K. T. Mursi, B. Thapaliya, Y. Zhuang, A. O. Aseeri, and M. S. Alkatheiri, "A fast deep learning method for security vulnerability study of XOR PUFs", *Electronics*, vol. 9, *10*, p. 1715, Oct. 2020, Number: 10 Publisher: Multidisciplinary Digital Publishing Institute. DOI: 10.3390/electronics9101715. [Online]. Available: https://www.mdpi.com/2079-9292/9/10/1715 (visited on 03/07/2021).

[7] M. Khalafalla and C. Gebotys, "PUFs deep attacks: enhanced modeling attacks using deep learning techniques to break the security of double arbiter PUFs", *in 2019 Design, Automation & Test in Europe Conference & Exhibition* (*DATE*), Florence, Italy: IEEE, Mar. 2019, pp. 204–209, ISBN: 978-3-9819263-2-3. DOI: 10.23919/

DATE.2019.8714862. [Online]. Available: https://ieeexplore.ieee.org/document/8714862/ (visited on 03/07/2021).

[8] A. A. Pour, V. Beroulle, B. Cambou, *et al.*, "PUF enrollment and life cycle management: solutions and perspectives for the test community", *in 2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–10. DOI: 10.1109/ETS48528.2020.9131578.

[9] S. Dongaonkar, S. P. Mudanai, and M. D. Giles, "From process corners to statistical circuit design methodology: opportunities and challenges", *IEEE Transactions on Electron Devices*, vol. 66, pp. 19–27, 2019.

[10] Y. Hori, T. Yoshida, T. Katashita, and A. Satoh, "Quantitative and statistical performance evaluation of arbiter physical unclonable functions on fpgas", *in 2010 International conference on reconfigurable computing and FPGAs*, IEEE, 2010, pp. 298–303.

[11] A. Maiti, V. Gunreddy, and P. Schaumont, "A systematic method to evaluate and compare the performance of physical unclonable functions", *in Embedded systems design with FPGAs*, Springer, 2013, pp. 245–267.

[12] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions", *in Proceedings of the 9th ACM conference on Computer and communications security*, ser. CCS '02, New York, NY, USA: Association for Computing Machinery, Nov. 18, 2002, pp. 148–160, ISBN: 978-1-58113-612-8. DOI: 10.1145/586110.586132. (visited on 03/07/2021).

[13] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "Extracting secret keys from integrated circuits", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, *10*, pp. 1200–1205, 2005.

[14] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation", *in 2007 44th ACM/IEEE Design Automation Conference*, IEEE, 2007, pp. 9–14.

[15] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "A new mode of operation for arbiter puf to improve uniqueness on fpga", *in 2014 Federated Conference on Computer Science and Information Systems*, IEEE, 2014, pp. 871–878.

[16] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "The butterfly puf protecting ip on every fpga", *in 2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, IEEE, 2008, pp. 67–70.

[17] F. Kodỳtek and R. Lórencz, "A design of ring oscillator based puf on fpga", *in 2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, IEEE, 2015, pp. 37–42.

[18] P. Tuyls, G.-J. Schrijen, B. Škorić, J. v. Geloven, N. Verhaegh, and R. Wolters, "Read-proof hardware from protective coatings", *in International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2006, pp. 369–383.

[19] C. Böhm, M. Hofer, and W. Pribyl, "A microcontroller sram-puf", *in 2011 5th International Conference on Network and System Security*, IEEE, 2011, pp. 269–273.

[20] B. Cambou, *Addressable puf generators for database-free password management system*, 2018.

[21] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, and U. Rührmair, "The bistable ring puf: a new architecture for strong physical unclonable functions", *in 2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, IEEE, 2011, pp. 134–141.

[22] B. Škorić, P. Tuyls, and W. Ophey, "Robust key extraction from physical uncloneable functions", *in International Conference on Applied Cryptography and Network Security*, Springer, 2005, pp. 407–422.

[23] W. Che, F. Saqib, and J. Plusquellic, "Puf-based authentication", *in 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 337–344. DOI: 10.1109/ICCAD.2015.7372589.

[24] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper data algorithms for puf-based key generation: overview and analysis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, *6*, pp. 889–902, 2015. DOI: 10.1109/TCAD.2014.2370531.

[25] R. E. Wright, "Logistic regression.", 1995.

[26] U. Ruhrmair, F. Sehnke, J. S olter, G. Dror, S. Devadas, and J. u. Schmidhuber, "Modeling attacks on physical unclonable functions", *in Proceedings of the 17th ACM conference on Computer and communications security - CCS '10*, Chicago, Illinois, USA: ACM Press, 2010, p. 237, ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866335. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1866307.1866335 (visited on 03/07/2021).

[27] W. S. Noble, "What is a support vector machine?", *Nature biotechnology*, vol. 24, *12*, pp. 1565–1567, 2006.

[28] Y. Cao, X. Zhao, W. Ye, Q. Han, and X. Pan, "A compact and low power ro puf with high resilience to the em side-channel attack and the svm modelling attack of wireless sensor networks", *Sensors*, vol. 18, *2*, p. 322, 2018.

[29] S. S. Zalivaka, A. A. Ivaniuk, and C.-H. Chang, "Low-cost fortification of arbiter puf against modeling attack", *in 2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4. DOI: 10.1109/ISCAS.2017.8050671.

[30] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu, "Machine learning resistant strong puf: possible or a pipe dream?", *in 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016, pp. 19–24. DOI: 10.1109/HST.2016.7495550.

[31] S. Kumar and M. Niamat, "Machine learning based modeling attacks on a configurable puf", *in NAECON 2018 - IEEE National Aerospace and Electronics Conference*, 2018, pp. 169–173. DOI: 10.1109/NAECON.2018.8556818.

[32] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, "An introduction to decision tree modeling", *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 18, *6*, pp. 275–285, 2004.

[33] T. Shi and S. Horvath, "Unsupervised learning with random forest predictors", *Journal of Computational and Graphical Statistics*, vol. 15, *1*, pp. 118–138, 2006.

[34] A. Ashtari, A. Shabani, and B. Alizadeh, "A new rf-puf based authentication of internet of things using random forest classification", *in 2019 16th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, 2019, pp. 21–26. DOI: 10.1109/ISCISC48546.2019.8985161.

[35] T. Kroeger, W. Cheng, S. Guilley, J.-L. Danger, and N. Karimi, "Effect of aging on puf modeling attacks based on power side-channel observations", *in 2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 454–459. DOI: 10.23919/DATE48585.2020.9116428.

[36] K. Fukushima, Y. Souissi, S. Hidano, *et al.*, "Delay puf assessment method based on side-channel and modeling analyzes: the final piece of all-in-one assessment methodology", *in 2016 IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 201–207. DOI: 10.1109/TrustCom.2016.0064.

[37] Y. Tanaka, S. Bian, M. Hiromoto, and T. Sato, "Coin flipping puf: a novel puf with improved resistance against machine learning attacks", *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, *5*, pp. 602–606, 2018. DOI: 10.1109/TCSII.2018.2821267.

[38] F. Murtagh, "Multilayer perceptrons for classification and regression", *Neurocomputing*, vol. 2, *5-6*, pp. 183–197, 1991.

[39] U. Ruhrmair, F. Sehnke, J. S olter, G. Dror, S. Devadas, and J. u. Schmidhuber, "Modeling attacks on physical unclonable functions", *in Proceedings of the 17th ACM conference on Computer and communications security - CCS '10*, Chicago, Illinois, USA: ACM Press, 2010, p. 237, ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866335. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1866307.1866335 (visited on 03/07/2021).

[40] M. S. Alkatheiri and Y. Zhuang, "Towards fast and accurate machine learning attacks of feed-forward arbiter pufs", *in 2017 IEEE Conference on Dependable and Secure Computing*, 2017, pp. 181–187. DOI: 10.1109/DESEC.2017.8073845.

[41] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: a convolutional neural-network approach", *IEEE transactions on neural networks*, vol. 8, *1*, pp. 98–113, 1997.

[42] M. Yue, N. Karimian, W. Yan, N. A. Anagnostopoulos, and F. Tehranipoor, "Dram-based authentication using deep convolutional neural networks", *IEEE Consumer Electronics Magazine*, vol. 10, *4*, pp. 8–17, 2020.

[43] M. Majzoobi, F. Koushanfar, and S. Devadas, "Fpga puf using programmable delay lines", *in 2010 IEEE international workshop on information forensics and security*, IEEE, 2010, pp. 1–6.

[44] A. Ali-Pour, D. Hely, V. Beroulle, and G. Di Natale, "Strong puf enrollment with machine learning: a methodical approach", *Electronics*, vol. 11, *4*, p. 653, 2022.

[45] A. Ali-Pour, D. Hely, V. Beroulle, and G. D. Natale, "Sub-space modeling: an enrollment solution for xor arbiter puf using machine learning", *in 2022 23rd International Symposium on Quality Electronic Design* (*ISQED*), 2022, pp. 1–1. DOI: 10.1109/ISQED54688.2022.9806267.

[46] A. Ali-Pour, D. Hely, V. Beroulle, and G. Di Natale, "Elaborating on sub-space modeling as an enrollment solution for strong puf", *in 2022 18th International Conference on Distributed Computing in Sensor Systems* (*DCOSS*), 2022, pp. 394–399. DOI: 10.1109/DCOSS54816.2022.00069.

[47] A. Ali-Pour, D. Hely, V. Beroulle, and G. Di Natale, "An efficient approach to model strong puf with multi-layer perceptron using transfer learning", *in 2022 23rd International Symposium on Quality Electronic Design* (*ISQED*), 2022, pp. 1–6. DOI: 10.1109/ISQED54688.2022.9806257.

[48] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, "Slender puf protocol: a lightweight, robust, and secure authentication by substring matching", *in 2012 IEEE Symposium on Security and Privacy Workshops*, 2012, pp. 33–44. DOI: 10.1109/SPW.2012.30.

[49] T. Idriss and M. Bayoumi, "Lightweight highly secure puf protocol for mutual authentication and secret message exchange", *in 2017 IEEE International Conference on RFID Technology & Application* (*RFID-TA*), IEEE, 2017, pp. 214–219.

[50] M. S. E. Quadir and J. A. Chandy, "Embedded systems authentication and encryption using strong puf modeling", *in 2020 IEEE International Conference on Consumer Electronics* (*ICCE*), 2020, pp. 1–6. DOI: 10.1109/ICCE46568.2020.9043104.

[51] http://www.pcp.in.tum.de/code/lr.zip.

[52] A. Maiti, V. Gunreddy, and P. Schaumont, "A systematic method to evaluate and compare the performance of physical unclonable functions", *in Embedded Systems Design with FPGAs*, P. Athanas, D. Pnevmatikatos, and N. Sklavos, Eds., New York, NY: Springer, 2013, pp. 245–267, ISBN: 978-1-4614-1362-2. DOI: 10.1007/978-1-4614-1362-2_11. (visited on 03/07/2021).

[53] Y. Hori, T. Yoshida, T. Katashita, and A. Satoh, "Quantitative and statistical performance evaluation of arbiter physical unclonable functions on FPGAs", *in 2010 International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico: IEEE, Dec. 2010, pp. 298–303, ISBN: 978-1-4244-9523-8. DOI: 10.1109/ReConFig.2010.24. [Online]. Available: http://ieeexplore.ieee.org/document/5695322/ (visited on 03/07/2021).

[54] K. T. Mursi, Y. Zhuang, M. S. Alkatheiri, and A. O. Aseeri, "Extensive examination of XOR arbiter PUFs as security primitives for resource-constrained IoT devices", *in 2019 17th International Conference on Privacy, Security and Trust (PST)*, Fredericton, NB, Canada: IEEE, Aug. 2019, pp. 1–9, ISBN: 978-1-72813-265-5. DOI: 10.1109/PST47121.2019.8949070. [Online]. Available: https://ieeexplore.ieee.org/document/8949070/ (visited on 03/07/2021).

[55] J. Tobisch and G. T. Becker, "On the scaling of machine learning attacks on pufs with application to noise bifurcation", *in Radio Frequency Identification*, S. Mangard and P. Schaumont, Eds., Cham: Springer International Publishing, 2015, pp. 17–31, ISBN: 978-3-319-24837-0.

[56] A. O. Aseeri, Y. Zhuang, and M. S. Alkatheiri, "A machine learning-based security vulnerability study on xor pufs for resource-constraint internet of things", *in 2018 IEEE International Congress on Internet of Things (ICIOT)*, 2018, pp. 49–56. DOI: 10.1109/ICIOT.2018.00014.

[57] T. A. Idriss, H. A. Idriss, and M. A. Bayoumi, "A lightweight puf-based authentication protocol using secret pattern recognition for constrained iot devices", *IEEE Access*, vol. 9, pp. 80 546–80 558, 2021.

[58] J. Tobisch and G. T. Becker, "On the scaling of machine learning attacks on pufs with application to noise bifurcation", *in Radio Frequency Identification*, S. Mangard and P. Schaumont, Eds., Cham: Springer International Publishing, 2015, pp. 17–31, ISBN: 978-3-319-24837-0.

[59] Q. Wang, O. Aramoon, P. Qiu, and G. Qu, "Efficient transfer learning on modeling physical unclonable functions", *in 2020 21st International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, USA: IEEE, Mar. 2020, pp. 1–6, ISBN: 978-1-72814-207-4. DOI: 10.1109/ISQED48828.2020.9137057. [Online]. Available: https://ieeexplore.ieee.org/document/9137057/ (visited on 03/07/2021).

[60] Z. Lyu, A. ElSaid, J. Karns, M. Mkaouer, and T. Desell, "An experimental study of weight initialization and lamarckian inheritance on neuroevolution", *EvoApplications Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany*, vol. 12694, 2021.

[61] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning", *in Artificial Neural Networks and Machine Learning – ICANN 2018*, V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, Eds., Cham: Springer International Publishing, 2018, pp. 270–279, ISBN: 978-3-030-01424-7.

[62] F. Ganji, D. Forte, and J.-P. Seifert, "Pufmeter a property testing tool for assessing the robustness of physically unclonable functions to machine learning attacks", *IEEE Access*, vol. 7, pp. 122 513–122 521, 2019. DOI: 10.1109/ACCESS.2019.2938408.

[63] P. Santikellur and R. S. Chakraborty, "A computationally efficient tensor regression network-based modeling attack on xor arbiter puf and its variants", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, *6*, pp. 1197–1206, 2020.

[64] N. Wisiol, K. T. Mursi, J.-P. Seifert, and Y. Zhuang, "Neural-network-based modeling attacks on xor arbiter pufs revisited", *Cryptology ePrint Archive*, 2021.

[65] A. Ali Pour, F. Afghah, D. Hély, *et al.*, "Helper data masking for physically unclonable function-based key generation algorithms", *IEEE Access*, vol. 10, pp. 40 150–40 164, 2022. DOI: 10.1109/ACCESS.2022.3165284.

[66] A. A. Pour, F. Afghah, D. Hely, V. Beroulle, and G. Di Natale, "Secure puf-based authentication and key exchange protocol using machine learning", *in IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2022)*, IEEE, 2022.

[67] G. T. Becker, "Robust fuzzy extractors and helper data manipulation attacks revisited: theory vs practice", *IEEE Transactions on Dependable and Secure Computing*, vol. 16, *5*, pp. 783–795, 2017.

[68] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper data algorithms for PUF-based key generation: overview and analysis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, *6*, pp. 889–902, 2015, ISSN: 1937-4151. DOI: 10.1109/TCAD.2014.2370531.

[69] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs", *in Cryptographic Hardware and Embedded Systems - CHES 2009*, C. Clavier and K. Gaj, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2009, pp. 332–347, ISBN: 978-3-642-04138-9. DOI: 10.1007/978-3-642-04138-9_24.

[70] R. Maes, A. Van Herrewege, and I. Verbauwhede, "PUFKY: a fully functional PUF-based cryptographic key generator", *in Cryptographic Hardware and Embedded Systems – CHES 2012*, E. Prouff and P. Schaumont, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2012, pp. 302–319, ISBN: 978-3-642-33027-8. DOI: 10.1007/978-3-642-33027-8_18.

[71] S. Puchinger, S. Mueelich, M. Bossert, M. Hiller, and G. Sigl, "On error correction for physical unclonable functions", *in SCC 2015; 10th International ITG Conference on Systems, Communications and Coding*, 2015, pp. 1–6.

[72] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs", *in Cryptographic Hardware and Embedded Systems – CHES 2008*, E. Oswald and P. Rohatgi, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2008, pp. 181–197, ISBN: 978-3-540-85053-3. DOI: 10.1007/978-3-540-85053-3_12.

[73] V. v. d. Leest, B. Preneel, and E. v. d. Sluis, "Soft decision error correction for compact memory-based PUFs using a single enrollment", *in International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2012, pp. 268–282.

[74] R. Maes, P. Tuyls, and I. Verbauwhede, "A soft decision helper data algorithm for SRAM PUFs", *in 2009 IEEE international symposium on information theory*, IEEE, 2009, pp. 2101–2105.

[75] S. Puchinger, S. Müelich, M. Bossert, M. Hiller, and G. Sigl, "On error correction for physical unclonable functions", *in SCC 2015; 10th International ITG Conference on Systems, Communications and Coding*, VDE, 2015, pp. 1–6.

[76] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs", *in International workshop on cryptographic hardware and embedded systems*, Springer, 2009, pp. 332–347.

[77] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs", *in International workshop on cryptographic hardware and embedded systems*, Springer, 2008, pp. 181–197.

[78] A. R. Korenda, F. Afghah, B. Cambou, and C. Philabaum, "A proof of concept sram-based physically unclonable function (*PUF*) key generation mechanism for iot devices", *in 2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking* (*SECON*), 2019, pp. 1–8.

[79] Y. Gao, Y. Su, W. Yang, S. Chen, S. Nepal, and D. C. Ranasinghe, "Building secure SRAM PUF key generators on resource constrained devices", *in 2019 IEEE International Conference on Pervasive Computing and Communications Workshops* (*PerCom Workshops*), IEEE, 2019, pp. 912–917.

[80] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper data algorithms for PUF-based key generation: overview and analysis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, *6*, pp. 889–902, 2014.

[81] D. Merli, F. Stumpf, and G. Sigl, "Protecting PUF error correction by codeword masking", *Cryptology ePrint Archive*, 2013.

[82] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith, "Secure remote authentication using biometric data", *in Advances in Cryptology – EUROCRYPT 2005*, R. Cramer, Ed., vol. 3494, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 147–163, ISBN: 978-3-540-25910-7 978-3-540-32055-5. DOI: 10.1007/11426639_9. (visited on 07/13/2020).

[83] Z. Paral and S. Devadas, "Reliable and efficient PUF-based key generation using pattern matching", *in 2011 IEEE international symposium on hardware-oriented security and trust*, IEEE, 2011, pp. 128–133.

[84] M. Hiller, M. Weiner, L. Rodrigues Lima, M. Birkner, and G. Sigl, "Breaking through fixed PUF block limitations with differential sequence coding and convolutional codes", *in Proceedings of the 3rd international workshop on Trustworthy embedded devices*, 2013, pp. 43–54.

[85] T. Idriss and M. Bayoumi, "Lightweight highly secure puf protocol for mutual authentication and secret message exchange", *in 2017 IEEE International Confer-*

ence on RFID Technology Application (*RFID-TA*), 2017, pp. 214–219. DOI: 10.1109/RFID-TA.2017.8098893.

[86] A. Ali-Pour, D. Hely, V. Beroulle, and G. Di Natale, "Strong puf enrollment with machine learning: a methodical approach", *Electronics*, vol. 11, *4*, 2022, ISSN: 2079-9292. DOI: 10.3390/electronics11040653. [Online]. Available: https://www.mdpi.com/2079-9292/11/4/653.

[87] A. Ali Pour, D. Hely, V. Beroulle, and G. Di Natale, "An Efficient Approach to Model Strong PUF with Multi-Layer Perceptron using Transfer Learning", *in International Symposium on Quality Electronic Design* (*ISQED 2022*), IEEE, Ed., Virtual event, United States: IEEE, Apr. 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03599336.

[88] A. R. Korenda, F. Afghah, and B. Cambou, "A secret key generation scheme for internet of things using ternary-states reram-based physical unclonable functions", *in 2018 14th International Wireless Communications & Mobile Computing Conference* (*IWCMC*), IEEE, 2018, pp. 1261–1266.

[89] X. Xi, A. Aysu, and M. Orshansky, "Fresh re-keying with strong pufs: a new approach to side-channel security", *in 2018 IEEE International Symposium on Hardware Oriented Security and Trust* (*HOST*), 2018, pp. 118–125. DOI: 10.1109/HST.2018.8383899.

[90] A. Alipour, A. Papadimitriou, V. Beroulle, E. Aerabi, and D. Hély, "On the performance of non-profiled differential deep learning attacks against an aes encryption algorithm protected using a correlated noise generation based hiding countermeasure", *in 2020 Design, Automation Test in Europe Conference Exhibition* (*DATE*), 2020, pp. 614–617. DOI: 10.23919/DATE48585.2020.9116387.

[91] C. Gu, C.-H. Chang, W. Liu, S. Yu, Y. Wang, and M. O'Neill, "A modeling attack resistant deception technique for securing lightweight-puf-based authentication", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, *6*, pp. 1183–1196, 2021. DOI: 10.1109/TCAD.2020.3036807.

[92] A. R. Korenda, F. Afghah, and B. Cambou, "A secret key generation scheme for internet of things using ternary-states ReRAM-based physical unclonable functions", *in 2018 14th International Wireless Communications Mobile Computing Conference* (*IWCMC*), 2018, pp. 1261–1266. DOI: 10.1109/IWCMC.2018.8450341.

[93] A. Alipour, D. Hely, V. Beroulle, and G. Di Natale, "Power of prediction: advantages of deep learning modeling as replacement for traditional PUF CRP enrollment", *in TrueDevice2020*, Grenoble, France, Mar. 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02954099 (visited on 03/07/2021).

[94] F. Regazzoni, S. Bhasin, A. A. Pour, *et al.*, "Machine learning and hardware security: challenges and opportunities -invited talk-", *in 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–6.

[95] L. Santiago, V. C. Patil, C. B. Prado, *et al.*, "Realizing strong puf from weak puf via neural computing", *in 2017 IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT)*, IEEE, 2017, pp. 1–6.

[96] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay, "A puf-based secure communication protocol for iot", *ACM Trans. Embed. Comput. Syst.*, vol. 16, *3*, Apr. 2017, ISSN: 1539-9087. DOI: 10.1145/3005715. [Online]. Available: https://doi.org/10.1145/3005715.

[97] A. Braeken, "Puf based authentication protocol for iot", *Symmetry*, vol. 10, *8*, 2018, ISSN: 2073-8994. DOI: 10.3390/sym10080352. [Online]. Available: https://www.mdpi.com/2073-8994/10/8/352.

[98] S. Garg, K. Kaur, G. Kaddoum, and K.-K. R. Choo, "Toward secure and provable authentication for internet of things: realizing industry 4.0", *IEEE Internet of Things Journal*, vol. 7, *5*, pp. 4598–4606, 2020. DOI: 10.1109/JIOT.2019.2942271.

[99] P. Mall, R. Amin, A. K. Das, M. T. Leung, and K.-K. R. Choo, "Puf-based authentication and key agreement protocols for iot, wsns, and smart grids: a comprehensive survey", *IEEE Internet of Things Journal*, vol. 9, *11*, pp. 8205–8228, 2022. DOI: 10.1109/JIOT.2022.3142084.

[100] M. Tahavori and F. Moazami, "Lightweight and secure puf-based authenticated key agreement scheme for smart grid", *Peer-to-Peer Networking and Applications*, vol. 13, *5*, pp. 1616–1628, 2020.

[101] M. H. Ameri, M. Delavar, and J. Mohajeri, "Provably secure and efficient puf-based broadcast authentication schemes for smart grid applications", *International Journal of Communication Systems*, vol. 32, *8*, e3935, 2019.

[102] M. N. Islam and S. Kundu, "Enabling ic traceability via blockchain pegged to embedded puf", *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, *3*, Apr. 2019, ISSN: 1084-4309. DOI: 10.1145/3315669. [Online]. Available: https://doi.org/10.1145/3315669.

[103] W. Wang, Q. Chen, Z. Yin, *et al.*, "Blockchain and puf-based lightweight authentication protocol for wireless medical sensor networks", *IEEE Internet of Things Journal*, vol. 9, *11*, pp. 8883–8891, 2022. DOI: 10.1109/JIOT.2021.3117762.

[104] V. K. V. V. Bathalapalli, S. P. Mohanty, E. Kougianos, V. P. Yanambaka, B. K. Baniya, and B. Rout, "A puf-based approach for sustainable cybersecurity in smart agriculture", *in 2021 19th OITS International Conference on Information Technology (OCIT)*, 2021, pp. 375–380. DOI: 10.1109/OCIT53463.2021.00080.

[105] P. K. Sadhu, V. P. Yanambaka, A. Abdelgawad, and K. Yelamarthi, "Performance analysis of ring oscillator puf for robust security in smart transportation", *in 2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, 2021, pp. 301–302. DOI: 10.1109/WF-IoT51360.2021.9596038.

[106] V. Pudi, S. Bodapati, S. Kumar, and A. Chattopadhyay, "Cyber security protocol for secure traffic monitoring systems using puf-based key management", *in 2020 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*, 2020, pp. 103–108. DOI: 10.1109/iSES50453.2020.00033.

[107] G. Bansal, N. Naren, and V. Chamola, "Rama: real-time automobile mutual authentication protocol using puf", *in 2020 International Conference on Information Networking (ICOIN)*, 2020, pp. 265–270. DOI: 10.1109/ICOIN48656.2020.9016538.

[108] Y. Xia, R. Qi, S. Ji, J. Shen, T. Miao, and H. Wang, "Puf-assisted lightweight group authentication and key agreement protocol in smart home", *Wireless Communications and Mobile Computing*, vol. 2022, 2022.

[109] E. Kordetoodeshki and S. Mirzakuchaki, "A new design for smart card security system based on puf technology", *International Journal of Machine Learning and Computing*, vol. 3, *3*, p. 267, 2013.

[110] Y. Chen, W. Kong, and X. Jiang, "Anti-synchronization and robust authentication for noisy puf-based smart card", *IEEE Access*, vol. 7, pp. 142 214–142 223, 2019.