— Master's Thesis —

Machine Learning Attacks on Majority Based Arbiter Physical Unclonable Functions

Manuel Oswald

Freie Universität Berlin

Department of Mathematics and Computer Science
Institute of Computer Science

supervised by
Prof. Dr. Marian Margraf
Prof. Dr.-Ing. Jochen Schiller

February 3, 2017

Abstract

Since secret keys are used as basis for several security features, e.g. encryption or authentication, there is a need to protect them from unauthorized access. Physical unclonable functions (PUFs) ought to fulfill this need with the possibility to derive secret keys from their intrinsic hardware imperfections These imperfections are linked inseparable with the device and can be easily used to derive secret keys from but are hard to read out. In addition, there is a high chance that they are unique to every instance. If that is true, they can be used to securely identify devices.

Nevertheless, certain PUFs can be successfully attacked by machine learning attacks. Machine learning attacks in combination with a feasible amount of challenge response pairs make it possible to create models of Arbiter PUFs and XOR Arbiter PUFs, which can be used to predict responses for unknown challenges. Large XOR Arbiter PUFs would prevent these attacks, but suffer instability.

To counteract these instabilities, the principle of majority vote is applied to Arbiter PUFs, which are used in XOR Arbiter PUFs. Its impacts on the stability of Arbiter PUFs and XOR Arbiter PUFs, and relevant attacks are verified by simulations. It is shown that majority vote enables large Majority XOR Arbiter PUFs. Furthermore, this work verifies that large Majority XOR Arbiter PUFs increase the complexity of non-invasive attacks exponentially by linear growth of the PUF. Thus, large Majority XOR Arbiter PUFs constitute a foundation for machine learning attack resistant PUFs. Since they are vulnerable to invasive attacks, they do not protect secret keys completely.

Abstract

Geheime Schlüssel sind die elementare Basis für Schutzziele der Informationssicherheit, wie z. B. Verschlüsselung oder Authentifizierung, und müssen aus diesem Grund vor Fremdzugriffen geschützt werden. Physical Unclonable Functions (PUFs) sollen diesen Schutz durch intrinsische Abweichungen ihrer Hardware, welche zur Ableitung von geheimen Schlüsseln genutzt werden können, bieten. Diese Abweichungen sind einfach zu nutzen, jedoch schwer auszulesen. Es besteht eine große Chance, dass die Abweichungen, die fest an die jeweiligen Geräte gekoppelt sind, eindeutig für eine Instanz des Gerätes sind. Trifft dies zu, ist eine eindeutige Identifizierung einzelner Geräte möglich.

Jedoch können bestimmte PUFs mittels Machine Learning Angriffen erfolgreich angegriffen werden. Machine Learning Angriffe nutzen Ein- und Ausgabepaare der zu angreifenden PUF, um ein Model dieser zu bilden. Das Model kann im Anschluss für das Vorhersagen von Ausgaben zu unbekannten Eingaben, der angegriffenen PUF, genutzt werden. Arbiter PUFs und kleine XOR Arbiter PUFs können somit erfolgreich angegriffen werden, wohingegen große XOR Arbiter PUFs resistent wären, aber durch Instabilität unbrauchbar sind.

Um diese Angriffe zu verhindern, wird das Prinzip Majority Vote in Kombination mit Arbiter PUFs, die für XOR Arbiter PUFs genutzt werden, angewandt. Sein Einfluss auf die Stabilität von Arbiter PUFs und XOR Arbiter PUFs sowie relevanten Angriffen wird mit Hilfe von Simulationen verifiziert. Es wird gezeigt, dass Realisierungen von großen Majority XOR Arbiter PUFs durch die Kombination von Majority Vote und Arbiter PUFs möglich sind. Darüber hinaus wird nachgewiesen, dass für große Majority XOR Arbiter PUFs die Komplexität der nicht invasiven Angriffe exponentiell steigt bei einem linearen Wachstum ihrer Größe. Aus diesem Grund können große Majority XOR Arbiter PUFs als Fundament für Machine Learning resistente PUFs dienen. Da sie dennoch anfällig für invasive Angriffe sind, ist es ihnen nicht möglich einen allumfassenden Schutz für geheime Schlüssel zu bieten.

Acknowledgments

I would like to express my great appreciation to Prof. Dr. Margraf for his valuable support in all aspects during the development of this thesis.

Special thanks should be given to Nils Wisiol and the members of the physical unclonable function (PUF) work group for all encouragement and inspiration.

I would also like to thank the Freie Universität Berlin and the HPC Team for giving access to the High Performance Computing resources.

Finally, I wish to thank my parents for their never ending support throughout my study and thank you to Laura Gindera, for all her love and understanding.

Berlin, February 3, 2017, Manuel Oswald

Contents

List of Abbreviations

ANN	artificial neural network	17
ASIC	application-specific integrated circuit	24
CLB	configurable logic block	24
CMA	covariance matrix adaptation	14
CMA-ES	covariance matrix adaptation (CMA) evolution strategy (ES)	14
CMOS	complementary metal–oxide–semiconductor	26
CNN	Cellular Nonlinear Networks	8
CPLD	complex programmable logic device	24
CRP	challenge response pair	4
CSA	cumulative step-size adaptation	14
DFA	deterministic finite automaton	32
ES	evolution strategy	12
FPGA	field-programmable gate array	24
HDL	hardware description language	24
IC	integrated circuit	7
LR	logistic regression	16
ML	machine learning	2
MUX	multiplexer	25
MV	majority vote	2
NXP	Next eXPerience	8
PAC	probably approximately correct	32
PUF	physical unclonable function	2
SR-latch	set-reset-latch	26
SVM	support vector machine	15
XOR	Exclusive Or	2

List of Figures

1 Introduction

The number of used electronic devices connected over the internet grew throughout the last two decades rapidly. Through the establishment of smartphones, numerous people are using services and transfer personal data via the internet. Apart from connected end-user devices, control units of industrial facilities or plants that provide all different kinds of supply can be operated via an internet connection.

These are just some examples for connections, where authentication and identification is needed to restrict the access to the qualified entity. Without restriction, every other entity has access and may be able to steal or cause damage otherwise. Since data, transferred via the internet, passes multiple nodes and can be eavesdropped, encryption is needed to protect the information contained. Authentication, identification, and encryption are just some security features needed to meet the requirements of information security.

The information security defines attributes a system has to meet to prevent it from threats, damage, and to minimize risks. The key concepts are confidentiality, integrity and availability. Data integrity means that data must not be changed without notice. All changes have to be comprehensible. Availability defines that data has to be available for a specified time frame. Confidentiality requires that data is not accessible to unauthorized entities. This applies all the time, even when data is transferred. Hence, concepts, for example authentication, are required to ensure the information security goals [28].

To meet the requirements of information security, secret keys are used. Secret keys have to be accessible by the devices to authenticate themselves and have to be protected. The attackers aim is to get the secret key to impersonate the entity the key belongs to. Different attacks to get to know the key are possible. Intrusion into a system that stores the key via a connection or with physical access to the key storage device extracting the key are two examples for possible attacks [48, 65].

Approaches like secret key storage devices, a memory specially protected against physical attacks, or one-way hash functions have been used so far to prevent those attacks [45]. A key, stored as hash of a one-way hash function, makes it harder for attackers to get to know the key. Knowing the secret key, used as input of a one-way hash function, makes it is easy to calculate its hash that is stored to represent the key. To verify a given key, its hash is calculated by the hash function and compared with the stored hash. If the hashes match, the key will be verified to be correct. When an attacker gets to know the hash, the key is still protected, since the one-way hash functions are hard to evaluate in the reverse direction with only the hash.

Yet due to the development of physical intrusion methods and the increasing performance of computer systems both techniques facing a number of challenges. For that reason physical unclonable functions (PUFs) have been introduced as a replacement for secret key memory. PUFs were also developed being an alternative to physical implementations of one-way functions used as authentication device such as smart cards [45]. It is suggested that PUFs offer a solution for security problems of the future with good prospects [70].

However, PUFs are also not immune to attacks, e.g. physical intrusion or machine learning attacks [4, 25, 57, 70]. Since PUFs are not completely studied, new types of PUFs and suitable attacks are proposed. This interplay between developing new PUFs that are claimed to be secure and developing of successful attacks leads to a huge variety of different PUFs [53].

Consequently, the Arbiter PUF and multiple modified versions based on the Arbiter PUF have been suggested. One of these versions is based on the idea of majority vote (MV) that has been suggested before, but has not been researched in detail [58]. This version of Arbiter PUFs is not publicly known yet, so we call it in this work the Majority Arbiter PUF. In combination with Exclusive Or (XOR) Arbiter PUFs it is then called Majority XOR Arbiter PUFs.

The concept of MV applied to Arbiter PUFs is introduced to overcome the problems XOR Arbiter PUFs face when growing large. This thesis studies the impact of MV on the possibility to build large XOR Arbiter PUFs. As there exist several successful attacks on Arbiter PUFs, their impacts on Majority Arbiter PUFs are unknown [17, 53]. Due to the modifications done to the Arbiter PUF, this thesis studies how existing attacks are affected by the use of MV.

The thesis is structured as follows: Chap. 2 gives an introduction to PUFs and machine learning (ML) attacks. A more detailed description of important ML attacks for this thesis is provided by Chap. 3. In Chap. 4 the Arbiter PUF specifications are outlined before in Chap. 5 applied attacks occurring in the current literature are shown. It is explained why only one of the attacks is relevant to the combination of XOR Arbiter PUFs and the idea of MV, which is introduced in Chap. 6.

To prove the possibility of large Majority XOR Arbiter PUFs and to research the impacts of MV on the attacks empirical, simulations of the PUFs and the attacks are built and described in Chap. 7. The results of the simulations are laid out in Chap. 8 and Chap. 9. Finally, a conclusion of the thesis summaries the results in Chap. 10.

2 Background

The idea of making an entity identifiable by its random variations goes far back in history. As an example a technique to identify documents and to prevent forgery was proposed in the year 1978 [9].

In the year 2001 a system has been proposed under the name physical unclonable function (PUF), based on a same idea [45]. Since that, PUFs have gained enormous research attention [4]. The idea is to use intrinsic device imperfections that are unique for every physical instance due to variations in the manufacturing process. These internal device differences makes them uniquely identifiable and can be used in multiple ways, as explained in this section.

This section shows the concept of PUFs and provides an overview about their classifications, applications, and different variations. Additionally, ML attacks, as one possible attack type for PUFs, are introduced. Both PUFs and ML attacks are needed to study the impacts of attacks on different PUFs in this thesis.

First, we describe the specifications of PUFs before we distinguish them by different classes in Sec. 2.1.1. Afterwards, current different PUFs and their applications are depicted. Finally, the ML attacks and one of their classifications are stated, to delimit the attacks part of this thesis.

2.1 Physical Unclonable Functions

A physical unclonable functions (PUFs) maps a set of challenges onto a set of responses. The combination of a challenge c and its response r is then called challenge response pair (CRP).

A challenge can be defined as sequence of n bits. The number of bits the sequence contains is called challenge length. The challenge length depends on the type of PUF and their implementation. Since there are different implementations of PUFs and no general definition, there is no restriction how many bits the response can contain.

Nevertheless, this work considers PUFs that response with a single bit. Hence, the function of a PUF can be expressed by the binary function $f : \{0,1\}^n \rightarrow \{0,1\}$, as long as environmental and random impacts that are occurring in PUF implementations and evaluation are not concerned.

So far there exists no general accepted complete formal definition of PUFs and their specifications [4]. However, multiple definitions are proposed, but no general consensus has been reached so far [3, 55, 72]. Yet some requirements must be met in order PUFs can fulfill their purpose, explained in Sec. 2.1.3. PUFs can be defined by the following three requirements:

- unclonability: As the name PUF encloses, one requirement for PUFs is being unclonable [70]. By unclonable it is meant that neither the producer of the PUF itself nor someone else can reproduce a PUF. Here reproducing means building a second instance with exact same binary mapping function. This is ensured by using the internal device differences, which occur in the manufacturing process and can not be controlled, to produce the PUF response.

- repeatability: Only PUFs that are repeatable can reproduce the same response for the same challenge. The reproduction of the same response is important, since it is the basis for all further processing when using PUFs, as shown in Sec. 2.1.3 [3].

- randomness: Randomness is important for the unpredictability of the PUF response. If one state of the binary response of the PUF is not been likely to occur with a likelihood of approximately 50 %, the response of the PUF can be predicted [10].

2.1.1 Classes of PUFs

This section gives an overview of the classifications of PUFs and how Arbiter PUFs can be classified. As there are different types of PUFs, they can be divided by different classifications.

The first classification divides them by their type of implementation. Accordingly, the sub-classes optical, electric digital, and electric analog are used. Electric implementations can be used directly in electronic devices and there has been an "extensive research in finding different electrical PUFs" [4]. In contrast to that, the first PUF implementation was an optical PUF.

Another classification separates different types of PUFs into strong and weak PUFs. Strong PUFs are defined by the following requirements [59]:

- The interface to evaluate challenges and to read out the corresponding response is freely accessible to everybody who holds the PUF.
- The number of unique challenges that can be evaluated by the strong PUF is very large. To evaluate challenges a finite evaluation time per challenge is needed. Consequently, it is impossible to read out all CRPs, even in a long time.
- The challenge response mapping behavior of strong PUFs is so complex that it is not possible to make any prediction on the response of a challenge. This also applies even if a large number of CRPs is known and used as source for prediction.

This definition of strong PUFs excludes PUFs that are classified as strong PUF, e.g. XOR Arbiter PUFs [4]. As no current available PUF fulfills these requirements, it has to be questioned if any PUF can meet these requirements. This pictures how premature PUF definitions are. Since definitions of PUFs are not a goal of this work, they will not be examined further.

In contrast to strong PUFs weak PUFs only have a few challenges, at least one [55]. Their responses are not meant to be given to the outside of the system and only used internally, e.g. as standard key [56]. A standard key does not change and can be used, as outlined in Chap. 1, for different purposes, e.g. for encryption in a Trusted Platform Module [67].

Weak PUFs provide an alternative to secret key storage devices, with the advantage that weak PUFs may be harder to read out [30]. Since weak PUFs do not have a large number of unique challenges it is easy to read out all their responses if free access is provided. As in this thesis PUFs, which provide a large amount of CRPs, are studied, weak PUFs will not be covered further.

Additional to strong PUF and weak PUF, controlled PUFs is the third classification. Controlled PUFs are based on a strong PUFs and a surrounding control logic that limits the access to the underlying PUF. This limiting logic is added to protect PUFs from attacks and can not be separated from the PUF [21, 58].

In contrast to the definition of strong PUFs, which is mentioned above, Arbiter PUFs are classified as strong PUF [54]. This thesis studies Arbiter PUFs and XOR Arbiter PUFs, which are both classified as strong PUF. Controlled PUFs will not be considered onward, as strong PUFs provide free access to the CRP interface in comparison to controlled PUFs.

We point out that there are further subclassifications, e.g. electrical PUFs can be divided into delay based PUFs and memory based PUFs [61].

2.1.2 Current Types of Strong PUFs

Different types of strong PUFs have been proposed. This section gives an overview about them and displays alternatives to the researched Arbiter PUF and XOR Arbiter PUF.

The optical PUF suggest by Puppu et al. uses a laser beamed through a transparent unit to project a pattern of light and dark spots onto a photo sensor [45]. The transparent unit contains reflecting particles, which interfere with the laser beam. By changing the angle between the laser source and the transparent unit the created pattern changes as well. The pattern is transformed by the photo sensor to a bit sequence and used to build the response. The PUF response is a representation of the pattern and the challenge is the angle that is used to apply the laser.

This is the first strong PUF, yet its application in the real world is difficult due to the laser and the photo sensor. Subsequently, several electric strong PUF have been suggested. One of them is the Power Grid PUF. The Power Grid PUF is based on the measurements of the variations of equivalent resistances of the integrated circuit (IC) [26].

Another important implementation is the Arbiter PUF, which is based on the runtime delay differences of two signals flowing through the IC [54]. These runtime delays differ because of variation in their circuit paths. Arbiter PUFs have been used as base for several other strong PUF constructions such as the XOR Arbiter PUF and will be studied in this thesis [4].

The Cellular Nonlinear Networks (CNN) PUF is an approach for an analog electric PUF where cells are interconnected to their neighbors in a two-dimensional net [13]. These interconnections have different strength that determines how signals are propagated through this net. After emitting a signal, it propagates through the net till the propagation stops and a response can be measured. In this way the response depends on a very large number of random electric components.

At last, the Crossbar PUF has to be mentioned. The Crossbar PUF has a very high independent information density and has a slow readout speed. This leads to a long readout time and is due to its physical construction, which makes it impossible to read out all responses. Trying to readout responses too fast destroys the circuit of the PUF and therewith the PUF itself. However, the slow readout speed can be a disadvantage depending on the application of the PUF [54].

2.1.3 Application of PUFs

PUFs can be used as basis for several security applications. In the field of encryption it is crucial to protect the secret key, since it is used for identification and authentication, as explained in Chap. 1. Everyone, who obtains the secret key, can impersonate the instance the key belongs to for example in cases where authentication is required.

Currently one method to store the secret key is using nonvolatile memory. However, it is possible with invasive and noninvasive physical tampering methods, e.g. micro-probing, to extract the stored secret key [30]. Hence, PUFs provide another possibility to include keys into an IC. One way is to keep the PUF responses private and derived secret key from it. This key then can be used for different propose, such as encryption [70].

Also, hardware fingerprinting can be done through the PUF responses, as the PUF is directly coupled with the device itself. With an included PUF it can be determined if any device is the device it claims to be. Beside authentication this can be used to detect fake ICs, which are distributed by forger, and to block them out [33]. Despite the limited study of PUFs to date, Next eXPerience (NXP) and Microsemi, two semiconductor manufacturers, already use PUF based key storage in some of their products [4].

2.2 Machine Learning Attacks

As this thesis verifies the success of machine learning (ML) attacks, this section provides some definitions for ML algorithms and gives an introduction for the interaction with PUFs. Afterwards, different learning approaches are named to clarify the type of attacks, which are used in this thesis to attack PUFs.

The field of ML has several approached and therefore multiple definitions [60]. An early definition was made by Arthur Samuel in 1959, which says:

"[Machine Learning is the field of study] that gives computers the ability to learn without being explicitly programmed." [62]

This definition has only a few restriction compared to the more formal definition of Tom Mitchell that was proposed in 1997 and says:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [42]

Mitchell defines that the ML algorithm, what he calls a computer program, learns from experience E and as a consequence improves its performance P on task E. However, both definitions claim that the ML algorithm learns, which is main the purpose of the ML algorithm. Samuel defines that the computer, which is represented by an algorithm here, "learns without being explicitly programmed." Mitchell in contrast does not define the experience E and how it is provided to the algorithm. Further, neither Samuel nor Mitchell mentions the learned knowledge of the algorithm. The idea of machine learning (ML) can also be defined by the ability of a system to produce knowledge from experience.

To realize this behavior the system uses an algorithm called ML algorithm. A ML algorithm starts with no knowledge or random given knowledge and modifies this knowledge. The modifications are based on given examples that provide the experience. This process is called training or learning. The modified knowledge of the ML algorithm is called model. This model, trained by the ML algorithm, can then make predictions about data it was trained with, new data, or both. It can recognize pattern or correlations and can classify data into different groups.

In the case of PUFs the term ML attack is used equivalently to ML algorithm. ML attacks are used to learn the response behavior of PUFs. To train the model by a ML attack CRPs, which are already know, are used. For challenges their responses are not known the model then can make predictions about whose responses.

2.2.1 Types of Learning Approaches

To categorize ML techniques by their approach the following three different learning approach classes are used [60]:

- supervised learning: The data includes input values and their desired output values for the ML algorithm. In the training process the ML algorithm has to make predictions about the data and in this way trains its model. The goal is to learn the rules how the input values mapped onto the output values.

- unsupervised learning: The data includes input values only. These data are used in the training process, so that the ML algorithm can find structures in the data. The goal can be to reveal that there is a structure in the data or to learn the structure. If the algorithm learned the structure, it will be able to repeat it to a certain level.

- semi-supervised learning: The data includes input values with their desired output values and input values without their output values. In the training process the ML algorithm has to structure the data and make already prediction for the data without output values.

- reinforcement learning: The ML algorithm is trained by reward and punishment to maximize its performance in a dynamic environment. This learning approach is based on a common way humans learn. The algorithm can choose an action to change the state of the environment over to another state. Depending on this transition the algorithm gets rewarded. The goal of the algorithm is to collect as much reward as possible.

There is a large variety of different ML techniques. Consequently, other categorizations, e.g. considering the desired output of the ML algorithm, exist. Yet the categorization by learning approaches is sufficient for this thesis. All ML attacks considered in this thesis are from the supervised learning class.

3 Machine Learning Attacks

Different ML attacks are used to attack PUFs. This section gives an overview of the techniques used in these attacks including the attack that is examined in this thesis. First, the required inputs for the ML attacks and the problem they are solving are explained. After that, each ML technique is outlined by its own.

The ML attack creates a PUF model of the attacked PUF, which acts as binary classifier. A binary classifier divides the samples, in the case of PUFs these are challenges, of a given set into two different classes 0 or 1 similar to the responses of a PUF. To build this model a training set $D = \{(z_1,e_1),...,(z_i,e_i)\}$ is necessary, whereas the integer i is the number of training set samples $d_i = (z_i,e_i)$ that are already classified by the attacked PUF. Let l be the number of input values of a sample d_i. This sample is then defined by its values $z_i = (z_{i,1},...,z_{i,l})$ and is classified into the class e_i.

With that training set the model can be trained by the ML algorithm. To evaluate the built model, a test set is created the same way and used to compare the output of the PUF model and the attacked PUF.

The most PUFs, known by the year 2014, which can be built by an electrical circuit, are related to internal values of the circuit [53]. These values determine the response for a given challenge. The goal of the ML attacks is to find a representation of these values, so that the created model classifies samples in the same way as the attacked PUF. How man samples of the test set the model classifies the same way as the PUF defines how precise the model has been trained. Accordingly, the values learned by the ML attack do not have to be exact the same values that defined the attacked PUF to classify samples of a set the same way as the PUF.

Whether a ML attack can find a binary classifier for a given set depends on the attack and if the set of samples is linear separable. Instead of a set of samples this also applies to binary functions $f : \mathbb{R}^n \to \{0,1\}$. One example for a not linear separable function is the XOR function. Some ML attacks, e.g. single-layer perceptron, can only learn linear separable sets and can therefore not learn XOR functions [41]. Two subsets $A \subseteq \mathbb{R}^n$, $B \subseteq \mathbb{R}^n$ are called linear separable if all their vectors $\vec{a} = (a_1,...,a_n) \in A$, $\vec{b} = (b_1,...,b_n) \in B$ can be separated by

$$\sum_{i=1}^{n} w_i a_i \leq w_{n+1} < \sum_{j=1}^{n} w_j b_j, \tag{3.1}$$

whereas $w_1,...,w_{n+1}$ are $n+1$ real numbers. The separating hyperplane is defined by the vector $\vec{x} = (x_1,...,x_n) \subseteq \mathbb{R}^n$ that satisfies $\sum_{i=1}^{n} w_i x_i = w_{n+1}$.In the case of building a classifier by the ML attacks the two subset A and B are subsets of the given set used to train the model. The ML attacks named in this chapter are common approaches to attack PUFs [53]. As there are different modifications and configurations used, only the ML algorithms itself are stated.

3.1 Evolution Strategies

The evolution strategy (ES) is an optimization technique that is inspired by the evolution process of the nature. In this process every species can adapt to the environmental conditions by changing its genes and only the fittest survive. From these fittest individuals the next generation will be built and inherits the attributes of their fittest parents genes.

This repeating process of classification by rating the fitness of every individual, inheriting the genes of the previous generation and modifying them leads to an approximation of the optimal solution for the specific environment. Every repetition of this evolutionary cycle counts as one generation and every individual represents a model with its own fitness value [4].

The model that is rated to be the fittest model after the last generation is the result of the algorithm and the best learned model by this time. It is also possible to approximate the at least optimal solution by letting the at least fittest survive. The procedure of the ES attack, in the case of finding the optimal solution, is displayed in the Alg. 1.

```
Algorithm 1: evolution strategy
Result: sets of models of the last generation
1 initialization of the models of the first generation
2 while termination criteria is not fulfilled do
3     modify all models
4     rate fitness of all models
5     sort all models by their fitness values
6     select number of the fittest models
7     create new generation from selected models
8 end
```

To initialize the models of the first generation in line 1, often random chosen normally distributed values are used. In the step of the lines 6 and 7 of the attack, it is determined how the number of models changes in the evolution process. There are options to define how many of the fittest models are selected and how many new models are created per generation.

Additionally, in the step of creating the next generation it can be defined if models of the old generation should be kept and if new randomly built models should be added. Adding new random models makes it possible for the attack to approximate a more optimal solution even if all models are already approximated to another solution [47]. Keeping old models ensures not to lose an individual of the previous generation, which outperforms the next generations models.

How the models are modified in the beginning of every generation in line 3 of the Alg. 1 is not specified. One way of modification is to add a random chosen value to all values of the individual. The fitness function in line 4 rates the fitness value of every individual and represents the above mentioned environmental conditions that the models get adapted to.

The sort function of line 5 sorts the results by their fitness values. The ES attack approximates an optimal solution if the models are sorted descending and vice versa. An attack ends when a termination criteria is fulfilled, which could be e.g. a maximum number of generations.

The ES attack is a randomized algorithm and therefore does not always determines. Nevertheless, it can approximate not linear separable problems and does not need a differentiable model [57].

3.2 CMA-ES

The covariance matrix adaptation (CMA) evolution strategy (ES) (CMA-ES) is based on the principles of the ES, which is shown in Sec. 3.1. Additional, the CMA-ES algorithm uses a covariance matrix to learn a second order model of the underlying function.

For this reason every value that defines the model and is learned by the algorithm has its own distribution, which will be optimized. Hence, the covariance matrix is used to represent pairwise dependencies between variables used in these distributions. Throughout the evolution process the mean values of the distributions, the covariance matrix and a step size value are adapted per generation.

The mean values, the covariance matrix, and the step size in combination with normal distributed randomly chosen values are used to create the model for the next generation in every evolution cycle [24]. To update the values used in the CMA-ES algorithm two principles are applied.

The first principle is the maximum-likelihood method. It is used to maximize the likelihood of previous models with the highest fitness values when updating the mean values. When updating the covariance matrix, the likelihood of previous search steps that lead to fitter models is increased.

The second principle captures two paths of the time progress of the distribution mean values. These two paths are called evolution paths. They are used to gaining information on correlations between consecutive steps of the mean values modification. The evolution paths becomes longer when consecutive step are taken in the same direction. The CMA process uses the first path to promote longer paths, which represent preferred directions of the distribution mean values modification, by the possibility of a faster distribution variance increase.

To control the step size value the second evolution path is used. The step size is adjusted by cumulative step-size adaptation (CSA) that reduces the importance of a step with the increasing number of taken steps [11]. This allows the algorithm to approximate the optimal solution fast and prevents the CMA-ES algorithm from to early adaptations.

The CMA is built to solve non-separable and ill-conditioned problems [24]. Problems are ill-conditioned when making little changes to their input lead to significant changes in their output [7].

3.3 Support Vector Machine

A support vector machine (SVM) is the model created by the SVM training algorithm. It classifies samples into two categories with dividing their multidimensional space by a hyperplane.

The number of dimensions of the space is defined by the number of values z of the samples. The model is trained the way that it divides the two categories by the largest possible gap. To achieve this, the training algorithm maximizes the distance from the hyperplane to the nearest samples of each category. If there exists such a hyperplane, known as maximum-margin hyperplane, it will have the larges margin to the closest samples of both categories.

Not all samples have to be considered when training the model. Samples that are located behind other samples of the same classification and are more far away of the hyperplane do not affect the location of the hyperplane, since they never become the closest samples. New samples are then predicted to belong to the same category as the samples that are on the side of the hyperplane where the new sample is located.

A complete separation into two categories by a linear hyperplane is only possible if the training data is linear separable. To make it possible to classify non-linear problems SVMs use the kernel method. In doing so, all training data and the z dimensional space is transformed to a space with a higher number of dimensions. All data transformed into a space with enough higher number of dimensions is linear separable. That can be even an infinite number of dimensions. The hyperplane is then determined in this higher dimensional space and transformed back to the original dimensional space where it represents a non-linear hyperplane. As these transformations are high in computational cost, the kernel method provides the possibility to define the hyperplane in a higher dimensional space and use it in the lower dimensional space without the transformations [12].

Another method used by SVM to be able to solve non-linear problems are slack variables. Slack variable allow the algorithm by undergoing punishment to classify samples in the training process wrong. This leads to a more flexible training process, reduces the number of needed samples and can build models of non-separable data.

3.4 Logistic Regression

The logistic regression (LR) is a regression analysis technique, which is used in statistic modeling to estimate relationships between variables. It is an approach to solve binary classification problems and is based on the logistic function $\frac{1}{1+e^{-z_{i,l}}}$.

The logistic function is a "S" shape and its values ranges between 0 and 1. The input value $z_{i,l}$ is transformed onto this range, since it is applied in the denominator of the function.

In the LR, additionally to the logistic function, all values $z_i = (z_{i,1}, ..., z_{i,l})$ of a sample that is going to be classified are combined with weights $\beta = (\beta_1, ..., \beta_l)$ and a bias β_0 is added. Hence, this modified version of the logistic function, known as LR model, is used to transform all values z_i by

$$\Pr[Y = 1|Z = z_i] = \frac{1}{1 + e^{-(\beta_0 + z_i^T \beta)}},$$

onto the range between 0 and 1. The value of the function is the likelihood $\Pr[Y = 1|Z = z_i]$ that a given sample classifies 1 under the condition its values are z_i. To calculate the binary classification, the likelihood is then divided into the classes 0 for $\Pr[Y = 1|Z = z_i] < 0.5$ and 1 for $\Pr[Y = 1|Z = z_i] \geq 0.5$.

The LR model is defined by the weights $\beta + \beta_0$. These weights must be estimated from the training data that includes already classified samples. A usual approach is the maximum likelihood estimation whereas the goal is that the likelihood, predicted by the LR model, is very close to 1 if the sample belongs to this class and the way round.

The maximum likelihood estimation does that by minimizing the error in the likelihood predicted by the model for the training data. For more detailed information I refer the reader to [2].

As a last point is has to be mentioned that the classification problems solved by LR do not need to be linearly separable but differentiable [57].

3.5 Perceptron

The perceptron is a supervised learning algorithm of binary classifiers and the simplest form of an artificial neural network (ANN) [50].

In case of a single-layer perceptron the values z_i of a sample d_i, which is going to be classified, are combined with the weights $\beta = (\beta_1, ..., \beta_l)$ and a bias β_0 is added, similar to LR shown in Sec. 3.4. However, the output value $f(z_i)$ of the perceptron is calculated by

$$f(z_i) = \begin{cases} 1 & \text{if } \beta \cdot z_i + \beta_0 > 0 \\ 0 & \text{otherwise} \end{cases}$$

and classifies the sample d_i in the classes 0 or 1. To obtain the weights and the bias, which define the perceptron model, the model is trained by the Alg. 2 and a training set with the samples d_i and their desired output e_i.

```
Algorithm 2: perceptron
Data: training set
Result: weights and bias
1 initialize the weights
2 initialize the bias
3 for each sample of the training set do
4 |   calculate the output of the perceptron
5 |   update the weights and the bias
6 end
```

The output value of the perceptron $f(z_i)$ is computed for a sample d_i with $\beta(p)$ and $\beta_0(p)$ at the time p in line 4. Afterwards, in line 5 the weights β and the bias β_0 are updated by

$$\begin{aligned} \beta_i(p+1) &= \beta_i(p) + \alpha \cdot (e_i - f(z_i)) \cdot z_i, \\ \beta_0(p+1) &= \beta_0(p) + \alpha \cdot (e_i - f(z_i)), \end{aligned}$$

whereas $\beta_i(p)$ are the weights β_i and $\beta_0(p)$ is the bias β_0 at the time p. A learning rate α is added that determines the value that is used to change the weights.

With these incremental updates the perceptron algorithm is an online learning algorithm. The algorithm is guaranteed to converge, and there is a limit for the maximum number of weight adjustments while trained under the condition that the training set is linear separable [50].

For a non-linear separable problem the perceptron algorithm does not terminate as it never reaches the optimal classification of all samples. A modification of the single-layer perceptron is the multilayer perceptron that is able to distinguish non-linear separable problems. For more detailed information about the multilayer perceptron I refer the reader to [51].

4 Arbiter PUFs

Arbiter PUFs were introduced already in 2002 as new delay-based PUF [19]. Over the year 2016 they got more attention, since their implementation is resource-efficient and low cost [5, 68]. However, there are still challenges due to the security concerns of classic Arbiter PUFs, as described in Sec. 5.1 [17, 53].

The relatively simple design, compared to e.g. cellular nonlinear network PUFs, of Arbiter PUFs makes it appealing to people to seek new improved possible varieties including the popular XOR Arbiter PUF, which will be presented in this section. Due to challenges, which occur when combining a large amount of Arbiter PUFs to a XOR Arbiter PUFs, it is so far impossible to realize large XOR Arbiter PUFs [52].

Furthermore, the success of attacks on XOR Arbiter PUFs depends on their size and as there is a limit it is also not possible to build XOR Arbiter PUFs that are secure and usable [18]. For these reasons there is still further improvement needed to call Arbiter PUFs to be practically secure, even for enhanced versions like for example XOR Arbiter PUFs. That is why I chose to study Arbiter PUFs and XOR Arbiter PUFs instead of other types of PUFs introduced in Sec. 2.1.2 in this thesis.

This sections starts with a description of how Arbiter PUFs work and the differences of XOR Arbiter PUFs. Subsequently, two models are explained. The first one gives an introduction on how Arbiter PUFs can be physically built. The second one defines the theoretical models of Arbiter PUFs used to attack them later in this thesis.

4.1 Arbiter PUFs

The concept of an Arbiter PUFs is based on the difference of the delays of two signals, which gets converted to a single bit response. To implement this concept, an Arbiter PUF has multiple connected stages and an arbiter at the end, as shown in Fig. 4.1.

Both signals propagate on parallel paths with the same layout length through all stages until they reach the arbiter. The arbiter then outputs the response bit depending on which input of the arbiter is triggered first by one of the signals. Every stage has three inputs, two for both signals and one for one bit of the challenge. This challenge bit determines if the signal paths will be crossed in this stage. If the bit is 0, both signals will flow parallel through the stage. If the bit is 1, the stage will swap the signal flow. So the paths for the signals through the stages exclusively depends on the challenge $c = (c_1, c_2, \cdots, c_n)$.

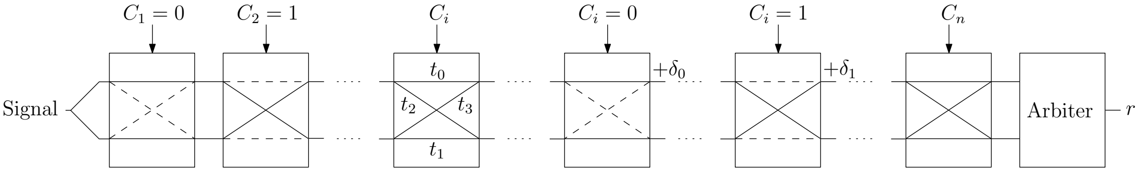

Figure 4.1: Arbiter PUF

The challenge bit length must match the number of stages in the Arbiter PUF so every stage gets a challenge bit assigned. This is called the size n of the Arbiter PUF in this work. The circuits of both paths are laid out to have logical equal length. But, due to the inaccuracies in the manufacturing process of the electronic devices used for the stages, which can not be influenced by the producer, transit times will differ from device to device. These differences can be seen as the intrinsic unique key of an instance of every Arbiter PUF. One stage for itself has 4 different signal paths:

- top input - top output: t_0
- bottom input - bottom output: t_1
- top input - bottom output: t_2
- bottom input - top output: t_3

Nevertheless, there are just two combinations possible, as the stage input is a Boolean value. Either its top input to top output and bottom output to bottom output or its top input to bottom output and bottom input to top output. These combinations lead to the delay differences $t_0 - t_1 = \delta_0$ and $t_2 - t_3 = \delta_1$ for the possible input 0 or 1 per stage. Additional, to the delay differences, which remain constant, deviations are occurring during the Arbiter PUF is queried. These deviations are noise and one way to model them is to use two distinct random variables. Both variables are normally distributed with mean zero and can be described as follows:

- At every stage of the Arbiter PUF, additionally to the chosen delay differences δ_0 or δ_1, normally distributed noise with zero mean and give variance occurs.

- At the arbiter normally distributed noise with zero mean and given variance leads to a different output of the comparison sometimes.

Due to the internal noise, the PUF may return different responses when given the same challenges twice. That is because challenges in combination with the different path delay values can result in a very small difference between the appearing signals on the arbiter. This small difference can be changed by the introduced noise or the arbiter can decide wrong due to its own noise every time the Arbiter PUF is queried and so the final response can be changed.

Hence, every challenge has its own stability value, which is defined in Eq. (4.4), and challenges with a lower stability value are called unstable. Whether a challenge is called unstable, depends on its stability value and the limit within a challenge is defined as stable. This limit is not generally defined and can be set as needed. Is the stability value below this limit the challenge is called unstable.

These are all parts to evaluate and to specify an Arbiter PUF. The XOR Arbiter PUF, which is based on Arbiter PUFs, will be outlined in Sec. 4.2. One way how an Arbiter PUF can be built and its possible internal structure will be presented in Sec. 4.3.1. The theoretical characterizations, a prerequisite for attacks, are part of Sec. 4.3.2. Successful attacks on Arbiter PUFs have been already developed and will be mentioned in Sec. 5.1.

4.2 XOR Arbiter PUFs

The XOR Arbiter PUF is one attempt to improve the security of Arbiter PUFs and has been proposed by Suh and Devadas in 2007 [68]. It combines multiple individual Arbiter PUFs with an Exclusive Or (XOR) function to a XOR Arbiter PUF, as shown in Fig. 4.2.

As the XOR function is not a linear separable function, as defined in Chap. 3, it adds non-linearity to the PUF. This non-linearity increases the effort on the side of the attacker and makes machine learning attacks more difficult [6, 30]. Another approach to add non-linearity are Feed-forward Arbiter PUFs proposed by Lim et al. [30].

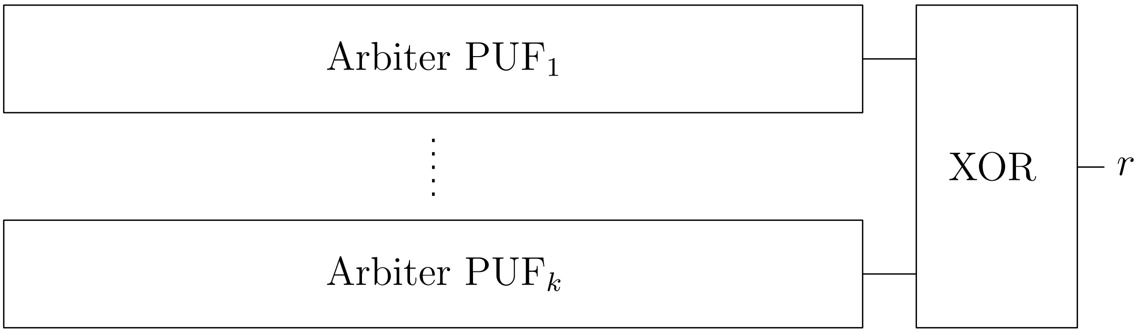

Figure 4.2: XOR Arbiter PUF

Every Arbiter PUF operates in the manner of the characterization in Sec. 4.1. To improve the security the XOR function adds up all responses to hide the response of every single Arbiter PUF instance. If the attacker does not know the response of a single Arbiter PUF, he will not be able to separately attack every single Arbiter PUF of the XOR Arbiter PUF [4]. This protects the underling Arbiter PUFs against direct attacks and under the condition that there are no more information revealed, through e.g. side channel attacks, as mentioned in Sec. 5.1, the attacker has to model the complete XOR Arbiter PUF at once to be successful.

The number of combined Arbiter PUFs k has the following two impacts on the PUF. Using more Arbiter PUFs increases the resistance against modeling attacks, but also decreases the stability, which makes the PUF less reliable [4, 18]. Stability here means the likelihood that the Arbiter PUF responses equally to an Arbiter PUF with the same internal values but without noise when processing the same challenge under the same environmental conditions.

The decrease of the stability of the entire XOR Arbiter PUF caused by using more Arbiter PUFs is due the fact that the less stable challenges of every Arbiter PUF are accumulated by the XOR function [52]. That leads to a maximum in the number of used Arbiter PUFs for real applications, since they have to meet a certain stability for a reliable repeatability [52].

The stability of an Arbiter PUF designed on a 65 nm technology is supposed to be 96 % [52]. This relation between the production technology and the stability, as Rostami et al. mentioned, is not part of this work and will therefore not be further studied [52]. Using 12 Arbiter PUFs in a XOR Arbiter PUF leads to a total stability of approximately 50 %, as the noise of 4 % of every Arbiter PUF gets added up and is not usable, since the response is not stable reproducible, as Ganji et al. implied [18].

Yet Yu et al. suggested that the accumulation of the Arbiter PUF noise values to obtain the XOR Arbiter PUF noise value, done by Ganji et al. is not precise enough. Accordingly, the stability of a XOR Arbiter PUF with $k = 12$ is 67 % for an even lower stability of 95.7 % for the included Arbiter PUFs [75]. Considering that, the limit for size k of a XOR Arbiter PUF depends on the required stability and the stability of the used Arbiter PUFs.

The size of the challenge of a XOR Arbiter PUF is the same as the size n of the underlying Arbiter PUFs. All internal Arbiter PUFs have the same size. The challenge must be distributed to all internal Arbiter PUFs. The approach for XOR Arbiter PUFs is to apply the same challenge to every Arbiter PUF. For the reason to make machine learning attacks more difficult, it is suggested to apply different challenges to every Arbiter PUF.

However, applying a more large challenge of size $n \cdot k$ to the XOR Arbiter PUF and then distributing individual challenges of size n to every Arbiter PUF reopens possibilities to directly attack individual Arbiter PUFs by changing only their challenge fraction [4].

Another way is to shift the challenge depending on the size of the PUF and the number of Arbiter PUFs every time before it will be applied to the next Arbiter PUF. Thus, in the best case every challenge bit takes every place in the challenge once when getting applied to different Arbiter PUFs. It is so far unproved that each input bit of an Arbiter PUF has a different high likelihood to affect the response. But, by transforming the challenge in this way, the likelihood of every bit to affect the response becomes more balanced, which makes the PUF less predictable and more difficult to attack.

As this challenge transformation is part of Lightweight PUFs, it will not be part of this work. For more detailed information I refer the reader to Majzoobi et al. [37].

4.3 Model

An Arbiter PUF and so its intrinsic secret can be manifested in different ways. The following section explains a physical implementation of Arbiter PUFs to provide an insight of Arbiter PUF implementations for real purpose.

Furthermore, the theoretical models of Arbiter PUFs are defined, since they are used by the simulation of Arbiter PUFs. They serve also as representation for the results of attacks stated in Sec. 3. This representation is not unclonable, as the values of the theoretical model are exposed and can be duplicated.

4.3.1 Physical

For real world use cases Arbiter PUFs have to be built as circuits to include them into devices. There are different possible engineering techniques to realize them. An application-specific integrated circuit (ASIC), field-programmable gate array (FPGA) or complex programmable logic device (CPLD) are the most common ones [34, 36, 66, 70].

The solution provided by an ASIC fits the logical layout of an Arbiter PUF best and is therefore the most accurate one. Yet for development purpose this technique is very expensive. Every time the layout changes a new wafer as core of an ASIC has to be produced. This manufacturing process is very costly and mainly used for the final product. Consequently, ASIC are used when high quantities of chips is needed.

A cheaper way is to use a FPGA or CPLD. A FPGA is a programmable IC, which can be programmed over and over again with very little effort. It consists of an array of configurable logic blocks (CLBs), here called gates, connected through a mesh. The hardware description language (HDL) tells the FPGA network how to connect the gates to obtain the desired circuit behavior. CPLD follow a similar principle as the FPGA, but are smaller in terms of resources and have a less complex structure.

Nevertheless, FPGA suffer from different problems making it difficult to use them as secure Arbiter PUF implementation. When describing the circuit layout of the chip with HDL, its code has to be compiled before transferring. In the compilation process the compiler determines, which physical gates on the chip will be used for the circuit design due to the specific chip type and its interconnect structure. This defines the used circuit traces between the used gates and lead to asymmetric routes.

Asymmetries in routing means that the trace to connect two gates is different in its physical length to its parallel opponent. This can lead to bias in the mean of the distribution of the delay differences of an Arbiter PUF, explained in Sec. 4.1, making the responses predictable and introduces a lack of randomness [36, 43]. The variation of the traces can be even higher than the manufacturing variations of the stage implementations, so that the response is mainly decided by the routing delays instead of the different variation combination of the used stage paths [38].

To counter these problems a concept of multiple combined Arbiter PUF called Double Arbiter PUF has been suggested [32, 33]. The Arbiter PUFs, considered in this thesis, are implemented by ASIC, as there are all necessary electronic components placed and connected freely. Accordingly, Arbiter PUF implementations in an ASIC will be further studied.

For Arbiter PUF implementations in an ASIC all conductor paths have to have a symmetric layout with the same capacitive loads to prevent the named bias of the delay differences. To implement the stages and the arbiter different electric components are required [34]. For the hardware realization of a single stage two 2-to-1 multiplexer (MUX) are used, as suggested by Rührmair et al. [29, 58].

There are analog or digital MUX available. Their design differences lead to a greater delay in passing the signal to its output for digital MUX compared to analog ones [40]. Additional differences, e.g. clipping parts of a continuous signal when exceeding a specific voltage, do not apply here, since the Arbiter PUF has only sign signals [64].

The sign signal of an Arbiter PUF is the signal, which propagates through the stages of the Arbiter PUF, shown in Sec. 4.1. It has two definite states, high and low, whereby high triggers the arbiter. Other values then high and low are not definite values of the signal and not used.

Hence, both types of MUX can be used unless they get mixed. Fig. 4.3 shows a single analog complementary metal–oxide–semiconductor (CMOS) MUX and how two of them used to build a complete stage component. The signals s_1 and s_2 are connected to the output o_1 and o_2 depending on the challenge bit c_i. Additional, there are some buffers used in the ASIC design, which yet are not needed for understanding.

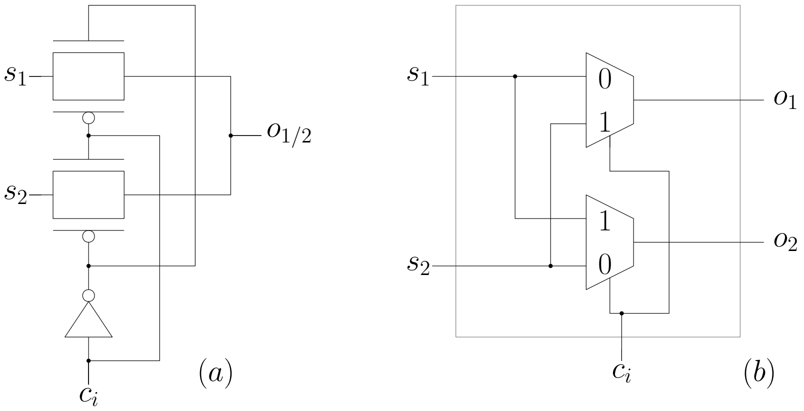

Figure 4.3: (a) Analog CMOS MUX realization with single output. (b) Two MUX connected to complete stage circuit.

The arbiter's task is to determine, which signal arrives first at the end of the stage chain. The ends of the stage chains are the same as the inputs of the arbiter. To provide this function a set-reset-latch (SR-latch) is used that can be realized in different ways with NOR, NAND, or AND-OR logic gates [40]. Using NAND logic gates is the method of choice, since it has a symmetric construction and because of that it is unbiased [31, 49].

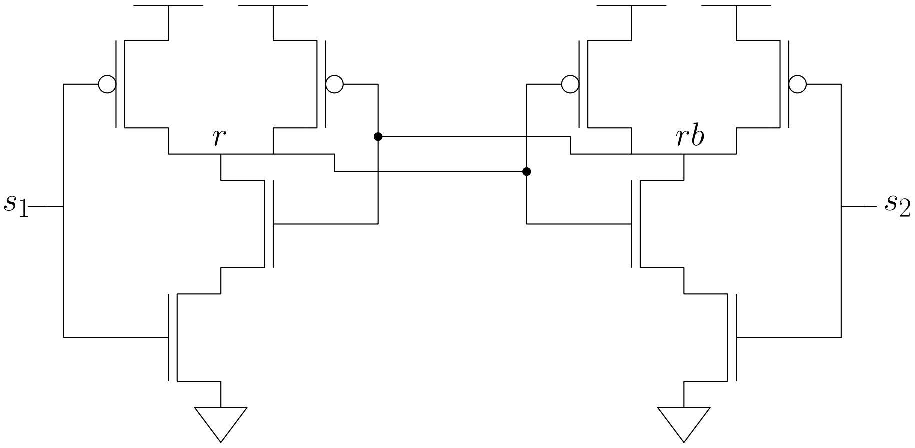

Figure 4.4: Arbiter switching circuit realized with CMOS NAND gates. The outputs r and rb start with the initial state high. If signal s_1 appears first, the output r will change to low, whereas rb stays high. If signal s_2 arrives first, it will behave the other way round.

Fig. 4.4 displays the switching circuit built with two cross-coupled CMOS NAND gates for an arbiter [40]. The internal states of the outputs r and rb are initial high. A rising edge of a signal occurring on s_1 will pull down r and the other way round. The output states low or high are equal to the response values of the PUF 0 or 1. Ensured by the cross-coupling, only one competing transition can win and the symmetric layout prevents circuit design based bias. NOR logic gates could be used likewise, as their SR-latch implementation is also symmetric.

All presented Arbiter PUF ASIC implementation properties are considered when creating the Arbiter PUF simulation in Sec. 7.1. This work does not examine physical Arbiter PUF implementations further, since they are not needed for understanding of ML attacks.

4.3.2 Theoretical

Every Arbiter PUF can be exactly represented by a theoretical model when all delay difference values of all stages and the noise distribution attributes are known. This section introduces the Arbiter PUF models needed for the simulation and the attack on them.

So far the challenge space consists of the values 0 and 1. For clearer notations and easier understanding that the challenge bit 1 flips the signal paths the challenge space will be converted from $\{0,1\}^n$ to $\{-1,1\}^n$ for the following equations. This can be done by $p(c_i) = (-1)^{c_i}$, which means 0 becomes 1 and 1 becomes -1.

To evaluate the response r of a challenge c of a specific PUF the sum of $\Delta D_{\text{Model}}(c)$ and the noise values $\Delta D_{\text{NoiseStage}}$ and $\Delta D_{\text{NoiseArbiter}}$ are computed. The final response r is then calculated by the signum function of the sum:

$$r = \text{sgn}(\Delta D_{\text{Model}}(c) + \Delta D_{\text{NoiseStage}} + \Delta D_{\text{NoiseArbiter}}). \tag{4.1}$$

The model $\Delta D_{\text{Model}}(c)$, which is the noise free Arbiter PUF representation, is the sum of all delay differences that are chosen to accord with the current challenge c. The sign of delay difference of every stage in the sum depends on the challenge bit of the current stage and all following stages. The current stage and every following stage could swap the signals and change the final response.

This is easier to understand by imagine an Arbiter PUF with just two stages where the delay difference values of the second stage are 0. The sign of the delay difference sum will be still definite by the challenge bit of the first and second stage. The second stage, according to its challenge bit, can always flip its stages paths independently of its delay difference values. A noise free Arbiter PUF model is thus be specified by

$$\Delta D_{\text{Model}}(c) = \sum_{i=1}^{n} \left(\delta_{c_i}^{(i)} \prod_{j=i}^{n} c_j \right), \tag{4.2}$$

for delay differences $\delta_{c_i}^{(i)}$ chosen by the bit of the challenge $c = (c_1, c_2, \cdots, c_n) \in \{-1, 1\}^n$ in the i-th stage of n stages.

As there are deviations in every component of the PUF when evaluating, noise needs to be modeled as well. Noise that occurs at the arbiter $\Delta D_{\text{NoiseArbiter}}$ can be modeled as a randomly drawn value of a normal distribution with zero mean and given variance $\sigma^2_{\text{ArbiterNoise}}$ depending only on the used electronic component. It is a common approach to use normally distributed random values for the path delay values and all noise values [14, 18, 57].

Noise that occurs at every path in every stage of the Arbiter PUF can be jointly drawn from a single normal distribution. This normal distribution has mean zero and variance scaling relative to the number of stages of the Arbiter PUF, as the sum of normally distributed stage noise is normally distributed itself. For that reason its random value is

$$\Delta D_{\text{NoiseStage}} \sim \mathcal{N}(m, \sqrt{n}\; \sigma_{\text{StageNoise}}), \tag{4.3}$$

with mean $m = 0$ and variance $\sigma^2_{StageNoise}$ per stage for n stages. Its standard deviation per stage adds up to the standard deviation of all stages $\sigma_{\text{AllStagesNoise}}$ of the PUF as follows:

$$\begin{aligned} \sigma_{\text{AllStagesNoise}} &= \sqrt{\sigma^2_{\text{AllStagesNoise}}} \\ &= \sqrt{n \cdot \sigma^2_{\text{StageNoise}}} \\ &= \sqrt{n} \cdot \sqrt{\sigma^2_{\text{StageNoise}}} = \sqrt{n} \cdot \sigma_{\text{StageNoise}}. \end{aligned}$$

The introduced noise leads to unstable challenges, as explained in Sec. 4.1. To define these unstable challenges precisely, we define the stability $\mathrm{Stab}(c)$ for challenge c to be

$$\Pr[\mathrm{sgn}(\Delta D_{\mathrm{Model}}(c) + \Delta D_{\mathrm{NoiseStage}} + \Delta D_{\mathrm{NoiseArbiter}}) = \mathrm{sgn}(\Delta D_{\mathrm{Model}}(c))]. \quad (4.4)$$

For a realistic model of an Arbiter PUF, noise has to added, as described. However, the noise free model $\Delta D_{\mathrm{Model}}(c)$ is the part with the hidden information, which an attacker want to expose. As widely common and used in attacks on Arbiter PUF, the noise free model can be specified in a case-distinction-free form and as linear function of dimension $n+1$ also called linear threshold function [4, 17, 20, 37, 39, 57]. An equal representation is used in this work to attack different types of Arbiter PUFs. The function is definite by

$$\Delta D_{\mathrm{Model}}(x) = \sum_{i=0}^{n+1} w_i x_i = \langle w, x \rangle, \quad (4.5)$$

whereas delay vector $w_i = (w_1, \cdots, w_{n+1})$ includes the delay differences $\delta_{c_i}^{(i)}$ of all stages and is calculated as follows:

$$\begin{aligned} w_1 &= \delta_0^{(1)} - \delta_1^{(1)}, \\ w_i &= \delta_0^{(i-1)} + \delta_1^{(i-1)} + \delta_0^{(i)} - \delta_1^{(i)} \text{ for } 2 \leq i \leq n, \\ w_{n+1} &= \delta_0^{(n)} + \delta_1^{(n)}. \end{aligned}$$

The feature vector x is computed from the challenge c by

$$\begin{aligned} x_i &= \prod_{i=1}^{n} c_i, \\ x_{n+1} &= 1. \end{aligned} \quad (4.6)$$

As in Sec. 4.2 introduced, XOR Arbiter PUFs include k multiple Arbiter PUFs whose single responses are combined by a XOR function to the final response. The XOR function is represented by multiplication in the $\{-1,1\}^n$ challenge space. Hence, we can model the response r of a XOR Arbiter PUF by

$$r = \prod_{i=1}^{k} \operatorname{sgn}(\Delta D^{i}_{\text{Model}}(x)) = \prod_{i=1}^{k} \operatorname{sgn}(\langle w_i, x \rangle)$$

for the noise free XOR Arbiter PUF model. Since the challenge c is the same for all Arbiter PUFs, the feature vector x stays the same as well. The feature vector is applied to all Arbiter PUFs and only the Arbiter PUF model w changes. Here introduced models are used in both to simulate attackable Arbiter PUF instances in Sec. 7.1 and in the attacks on them in Sec. 7.2.

5 Attacks

The aim of attacks on Arbiter PUF is to predict responses for challenges their responses are unknown. To accomplish this on the one hand it is possible to reveal the intrinsic secret represented by the inaccuracies of the Arbiter PUF circuit. On the other hand the challenge response behavior of the PUF can be adapted. Both methods can then be used to impersonate the attacked PUF.

This section first names attacks on Arbiter PUFs occurring in the current literature. It separates them into different classes of attacks to distinguish the attacks studied in this work from other attacks. Then attacks on XOR Arbiter PUFs and their success are mentioned. Finally, the relevant attacks for this thesis are named in addition to the reasons for their choice.

5.1 Attacks on Arbiter PUFs

Since Arbiter PUFs were one of the most promising PUF implementation, multiple attacks have been developed. One of the first attacks reveals the internal values of the Arbiter PUF by linear programming [44]. This approach solves a linear equation system after collecting enough CRPs, because an Arbiter PUF can be represented as linear threshold function, as shown in Sec. 4.3.2. For a good result the stability of the responses of the CRPs should be as high as possible.

As the response of every CRP includes some noise, explained in Sec. 4.3.1, one strategy of the attacker is to reduce the noise by using majority vote, i.e. evaluating the same challenge multiple times and picking the most appeared response. Therefore, the attacker has to be able to evaluate chosen challenges. Another option is using an algorithm to learn noisy linear threshold function, as Blum et al. showed can be done in polynomial time [8].

The linear programming attack is a non-invasive modeling attack where the attacker needs to collect CRPs to build a model of the PUF for example by a ML algorithm. Subsequently, the model can predict responses for new challenges, which have not been used to create the model itself. How precise these predictions are depends on the number of captured CRPs and used parameters to build the model. The result and the model building performance vary for different ML techniques. How the attacker gets to know the CRPs in a real use case depends on the way the PUF is used and the CRPs transmitted and is therefore not part of this work.

Arbiter PUFs are strong PUFs and their operation frequency is relatively high. For that reason they provide the option to freely access a large amount of CRPs [57]. On the contrary, some PUFs post-process their response in a protected chip environment to make the PUF response itself inaccessible to protect the PUF [20, 68]. All non-invasive attacks mentioned in this thesis investigate PUFs despite how they obtain CRPs. For this reason I assume to have free access to the PUF and to be able to measure CRPs without restrictions.

It has been shown by Rührmair et al. that Arbiter PUFs can be successfully attacked resulting in a model with even higher stability as the Arbiter PUF insilicon implementation has [57]. For these attacks he used SVMs, LR, and ES where LR performed best in terms of predicting randomly chosen challenges.

Other techniques to create accurate Arbiter PUF models are ANN and probably approximately correct (PAC) learning, whereas PAC learning is an analysis framework and not a ML technique. ANNs need a relative large amount of CRPs to achieve a high prediction accuracy of new challenges [27]. The approach for a PAC learning algorithm by Ganji et al. requires a deterministic finite automaton (DFA) based representation of Arbiter PUFs to attack them successfully [17].

All non-invasive attacks mentioned so far are based only on the values of the responses of the attacked PUF. A different attack is the reliability-based machine learning attack proposed by Becker [4]. It is based on the reliability of the PUF responses than on their values. This in combination with a CMA-ES algorithm makes it possible to create Arbiter PUF models with high accuracy. We count this attack as non-invasive attack, as it relies only on the response of the PUF and does not need physical access to the PUF like side channel attacks.

In addition to non-invasive modeling attacks, physical side channel attacks are possible. As mentioned before, in real attack scenarios, the PUF response or a large set of CRPs is may be not available to the attacker, so he has to obtain information differently. Physical side channel attacks require physical access to the PUF.

As an example, the photonic emission analysis attack, where the attacker measures the evolved photons of the bottom side of the Arbiter PUF chip to reveal all its internal delay values, has been proposed by Tajik et al. [70]. With these values, the attacker can create an exact model of the Arbiter PUF, as shown in Sec. 4.3.2. Different combinations of machine learning attacks and side channel attacks have been proposed as well [35, 74].

However, this thesis is focused only on non-invasive machine learning attacks, as they do not require physical access to the Arbiter PUF chip.

5.2 Attacks on XOR Arbiter PUFs

Many attacks on Arbiter PUFs can also be applied to XOR Arbiter PUFs. As main differences, the attacks additionally have to cope with the added non-linearity by the XOR function, the increasing number of Arbiter PUFs, and the fact that the responses of all Arbiter PUFs are not individually accessible.

The increasing number of Arbiter PUFs increases the complexity of an attack. This complexity grows in form of raising time consumption of the attack or a greater number of required CRPs. Additional, their growth rates are important factors to build secure XOR Arbiter PUFs that can not be successfully modeled. An exponential growing complexity to attack a XOR Arbiter PUF that can be created with linear growing effort can lead to a secure PUF.

Nevertheless, on one hand this complexity grows exponentially with the number of used Arbiter PUFs for the most attacks and on the other hand the size k of a XOR Arbiter PUF is limited by the growing total noise, as presented in Sec. 4.2 [57]. The reports about current effective attacks on XOR Arbiter PUFs differ in the maximum size k by 4 to 6 [18, 74].

Under these conditions Ganji et al. developed a PAC analysis algorithm to successfully model XOR Arbiter PUFs [18]. Using ANN and SVM proved to be rather inefficient when $k > 2$ [27]. Equal bad performed the ES attack. However, the LR algorithm is able to model XOR Arbiter PUFs of size $k \leq 6$ by $n = 64$ [57].

All attacks mentioned so far grow exponentially in their complexity for linear increasing k. Only the CMA-ES attack that is based on the reliability of the responses claims to have linear growing time consumption for rising k [4]. That means for example using double the amount of Arbiter PUFs in a XOR Arbiter PUFs leads only to twice the required CRPs to run a successful attack.

5.3 Relevant Attacks

The complexity of all attacks applied to XOR Arbiter PUFs grows exponentially with increasing number of Arbiter PUF, except of the reliability based CMA-ES attack, as outlined in Sec. 5.2. Becker points out that his approach to combine the reliability based CMA-ES attack with a normal CMA-ES attack "results in only a linear increase in needed number of challenge and responses for increasing numbers of XORs" [4].

A XOR Arbiter PUF with an arbitrary large number of Arbiter PUFs would prevent all attacks but the CMA-ES attacks. To overcome the problems, large XOR Arbiter PUFs encounter, the concept of MV has been developed, as shown in Chap. 6. MV claims to make it possible to build XOR Arbiter PUFs with large number of Arbiter PUFs, which will be studied in Chap. 8.

As a result, the only attacks that can be applied to large XOR Arbiter PUFs with a feasible required CRPs are CMA-ES attacks. Hence, the impact of using MV in combination with Arbiter PUFs on the success of the CMA-ES attacks will be presented in Chap. 9.

6 Majority Arbiter PUFs

Most of the attacks, named in Sec. 5.2, increase exponentially in their complexity for a growing number of used Arbiter PUFs in a XOR Arbiter PUF. Consequently, it can be claimed that arbitrary large XOR Arbiter PUFs would constitute a secure PUF in terms of all now known non-invasive modeling attacks except of the CMA-ES attack. This is possible because only linear enlargement of XOR Arbiter PUFs will lead to exponential growth in attack effort, even with the fastest known attack.

To overcome the stability problem of XOR Arbiter PUFs, mentioned in Sec. 4.2, which currently deter them from been enlarged we introduce the concept of MV [36]. The concept of MV has been already mentioned by Rührmair et al. though they did not further investigate its impact on the possibility to create large XOR Arbiter PUFs [58]. MV can also be used by the attacker as a technique to obtain responses with high stability of a PUF when required [17, 44]. As the attacks of this work are not based on MV, this purpose of MV is not part of this thesis.

This section introduces the concept of MV in combinations with Arbiter PUFs. Afterwards, it is outlined how this combination is used to build large XOR Arbiter PUFs. Finally, alternative ways to combine multiple Arbiter PUFs are mentioned.

6.1 Majority Based Arbiter PUFs

To apply MV to an Arbiter PUF a challenge is evaluated m times. Afterwards, the most occurring response is the final response for that challenge. We define the combination of MV and Arbiter PUF to be a Majority Arbiter PUF.

In this way the total number of unstable challenges, as explained in Sec. 4.1, gets reduced and the stability of the Arbiter PUF increases. Every challenge has a different stability, as defined in Eq. (4.4). These values vary between 50 % and 100 %. The closer the value is to 50 %, the more votes are necessary to get a more stable response for this challenge. Accordingly, how many votes are needed depends on the stability of the Arbiter PUF that is required. For a more detailed view on the relations between stability and the number of required votes, different combinations will be simulated and evaluated in Chap. 8.

6.2 Majority Based XOR Arbiter PUFs

To make XOR Arbiter PUFs secure against modeling attacks they have to use a large amount of Arbiter PUFs, which involves a decrease in their stability. To boost the stability of XOR Arbiter PUFs, MV is used for every single Arbiter PUF instead of the complete XOR Arbiter PUF, as shown in Fig. 6.1. A XOR Arbiter PUF containing Majority Arbiter PUFs instead of Arbiter PUFs is named Majority XOR Arbiter PUF.

The stability of the XOR Arbiter PUF increases with the increase of the stability of every Majority Arbiter PUF. As the stability decreases with higher *k*, this has to be counteracted with a larger amount of *m* MVs. A theoretical proof by Wisiol et al. has showed that a polynomial increase of the number of MV leads to an exponential growth of the stability of the Majority XOR Arbiter PUF [73].

In this work we verify by simulation how this principle can be used to create large Majority XOR Arbiter PUFs and how attacks are affected by that. A more detailed report on the relations between the stability values of the challenges for large Majority XOR Arbiter PUFs will be examined in Chap. 8.

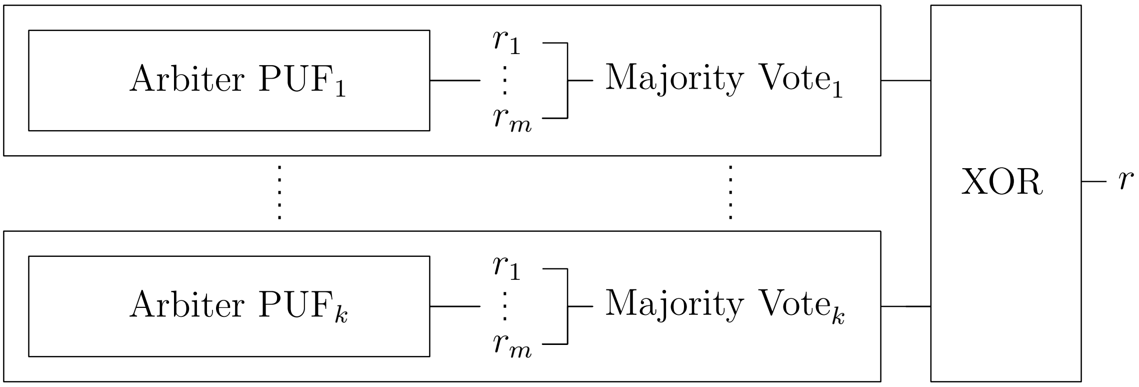

Figure 6.1: Majority XOR Arbiter PUF

6.3 Combiner Functions

Beside the XOR function other Boolean functions could be used to combine multiple Arbiter PUF. In addition to being a Boolean function it is important that the function obfuscates the single responses of the Arbiter PUFs. Otherwise, the Arbiter PUFs are directly vulnerable to modeling attacks on Arbiter PUFs.

Bent functions are one example for Boolean functions, which can be used instead of the XOR function [1, 16, 63]. They are highly non-linear, which is desired in the case of Arbiter PUFs, since they can be represented by a linear threshold functions. Attacks mentioned in Sec. 5.1 are able to successfully attack these linear threshold functions. The non-linearity increases the complexity of attacks, as described in Sec. 4.2. Beyond that Bent functions fulfill the strict avalanche criterion [15]. The strict avalanche criterion means if any number of input bits is flipped the output bit flips with a likelihood of 50 %.

How these attributes of Bent functions can help to prevent successful attacks and analysis of the underlying Arbiter PUFs has to be examined. This thesis is focused only on the XOR function used as combiner function and will not consider other possible combiner functions further.

7 Simulation Design

To verify the impact of MV on the stability of delay based PUFs, we simulate different versions of Arbiter PUFs combined with MV. Based on these simulations the influence of MV on the only relevant attack on large XOR Arbiter PUFs, as pointed out in Sec. 5.3, is studied.

Most parts of the implementation are written in Python. The simulations are carried out by desktop computers and High Performance Computing resources. All figures that are used to display the results are created by Wolfram Mathematica.

This section gives a more detailed explanation of the simulations and the attacks. Thus, it starts with a description of the simulation of Arbiter PUFs that is basis for all other PUF simulations in this thesis. Afterwards, the principles used by the different attacks on Majority Arbiter PUFs and Majority XOR Arbiter PUFs are pointed out.

7.1 PUF Design

The Arbiter PUF simulations are based on the Arbiter PUF representations, presented in Sec. 4.3.2.

The sum of independent random variable, regardless of their distribution, tend to toward a normal distribution, known as the central limit theorem [46]. Hence, it is a common approach for natural randomly occurring deviations, whose distribution is unknown, to be chosen of a normal distribution. For that reason, the delay difference values δ_i, which represent the variation of the Arbiter PUF due to the manufacturing process, are randomly chosen normally distributed values with zero mean and given variance.

The different noise values added in Eq. (4.1) are also normally distributed random values with zero mean and given variance. The variance of the stage noise distribution, named in Sec. 4.3.2, depends on the size n of the Arbiter PUF, as required by Eq. (4.3).

For a physical Arbiter PUF implementation, the distribution of the delay difference values and all noise values depend on the used electronic components, as shown in Sec. 4.3.1. The ratio between $\sigma_{\text{StageNoise}}$ and σ_{Model} and the value of $\sigma_{\text{ArbiterNoise}}$ can affect the stability of the challenges of the Arbiter PUF by promoting wrong PUF evaluations if the noise is relatively high. As this work studies the impact of adding MV to Arbiter PUFs in a comparison, the real value of the ratio does not matter. The impact of changing the ratio is not part of this work and could be the subject of another thesis. Yet it is crucial to not change the ratio during the simulation to obtain results that are comparable.

The response of the Arbiter PUF model is simulated by Eq. (4.2) or Eq. (4.5) before the noise values are added. All other implementations of PUF concepts used in this thesis, e.g. the Majority Arbiter PUF, are based on these Arbiter PUF equations. For the attack simulation of the CMA-ES attack Eq. (4.5) is used, since the delay vector w is approximated, as outlined in Sec. 7.2.

7.2 Machine Learning Design

To attack Majority Arbiter PUFs and Majority XOR Arbiter PUFs, which are introduced in Chap. 6, the reliability based CMA-ES attack introduced by Becker is interesting to study [4]. Further, sources have been used [4, 23, 24]. It is the only relevant attack that can successfully create well performing models of large XOR Arbiter PUFs, as termed in Sec. 5.3.

Since large XOR Arbiter PUFs face some stability challenges, the concept of MV is used to overcome them, as expressed in Sec. 6.2. Accordingly, the attack is applied to Arbiter PUFs and XOR Arbiter PUFs with and without the concept of MV to examine its impact on the attack. The attack implementation is based on the original source [4]. A model performs well if it correctly predicts a high proportion of the challenges from a test set.

The reliability based CMA-ES algorithm relies on the principles of the classic CMA-ES algorithm, which is described in Sec. 3.2. To provide a better understanding of how the attack works, this section outlines the applied principles and the attack implementations. First, the reliability approach, used in a fitness function, to rate the models, created by the algorithm, is explained. Subsequently, it is shown how this is applied in the CMA-ES algorithm to approximates well performing models of the attacked Arbiter PUF. Finally, it is clarified how to use the algorithm to successfully attack XOR Arbiter PUFs.

7.2.1 Reliability Based Fitness Function

The ES algorithm is inspired by the natural evolution process of survival of the fittest, stated in Sec. 3.1, and thus needs a method to score the fitness of a trained model. This function is called fitness function. In Sec. 7.2.2, we will outline how the fitness function is used in the CMA-ES algorithm.

Becker's fitness function uses reliability values of challenges that are evaluated by the attacked PUF. To measure the reliability value h_i, which is defined in the next paragraph, of a challenge, the challenge has to be evaluated multiple times by the attacked PUF. Therefore, the attack requires the possibility to freely evaluate challenges similar to a chosen-plaintext attack [48]. This process of measuring multiple challenge evaluations has to be done only once, since the same values are used during the complete attack. The set of multiple evaluated challenges is in the case of the reliability based CMA-ES attack the training set with v number of challenges. The test set is created by single evaluations of different challenges, as specified in Cap. 3.

How the definition of reliability differs from the definition of stability that is given in Eq. (4.4), in terms of the reliability based CMA-ES attack, is highlighted now. Let j be the number of times a challenge is evaluated by the same PUF and let r_i, $1 \leq i \leq j$ be the responses captured to calculate the reliability h_i. The reliability h_i of a challenge c is computed of its responses r_1,r_2,...,r_j as follows [4]:

$$h_i = |\frac{j}{2} - \sum_{l=1}^{j} r_l|. \tag{7.1}$$

In Eq. (7.1) it can be seen that the reliability value of a challenge c has its maximum of $\frac{j}{2}$ when the PUF evaluates with all r_i, $i \leq j$, are simultaneously 0 or 1, for all j times of evaluations. A challenge that evaluates one or more times of j evaluations to a different response as the other responses has a lowered reliability and is called unreliable challenge. Consequently, $h = (h_1,...,h_v)$ is the reliability vector of v challenges from the training set evaluated by the attacked PUF.

Becker's fitness function uses also the hypothetical reliability values of challenges evaluated by the modeled PUFs. The hypothetical reliability of a challenge has also a different definition than the reliability and the stability, in terms of the reliability based CMA-ES attack. To rate all models, hypothetical reliability values have to be calculated for every model created by the algorithm. The models are defined by their delay vectors w of real-valued numbers and of size $n+1$, since they are used in the Arbiter PUF model representation by Eq. (4.5) and are known by the algorithm. Additionally, the feature vector x_{n+1} of real-valued numbers, which is defined in Sec. 4.3.2, is computed from every challenge of the training set by Eq. (4.6). Hence, the vectors w, the feature vectors x, and the Arbiter PUF model Eq. (4.5) are used to calculate the hypothetical reliability g_i for a challenge as follows:

$$g_i = \begin{cases} 1, & \text{if } |\langle w, x \rangle| > \epsilon, \\ 0, & \text{if } |\langle w, x \rangle| < \epsilon. \end{cases} \tag{7.2}$$

The hypothetical reliability categorizes every challenge for the model defined by w into hypothetical reliable and hypothetical unreliable. A challenge is defined to be hypothetical reliable when its delay difference computed by $\langle w, x \rangle$ is greater than the error boundary ϵ and results in $g_i = 1$. The challenge is defined to be hypothetical unreliable when its delay difference is lower than the error boundary ϵ and results in $g_i = 0$. This relies on the facilitation of a low delay difference value being changed by noise, which results in a different sign of the value and thus a different response, as outlined in Sec. 4.1.

The error boundary ϵ is a flexible limit, which determines the separation, and is approximated in the attack. Its value is approximated through the covariance matrix adaptation similar to the way the model values are approximated, as presented in the next Sec. 7.2.2. The hypothetical reliability vector is $g = (g_1, ..., g_v)$ for v evaluated challenges from the training set by every created model of the attack.

As last step Becker's fitness function computes the Pearson correlation coefficient between the reliability values h_i and the hypothetical reliability values g_i of all challenges evaluated by the attacked PUF. The number of evaluated challenges by the attacked PUF is the size of the training set, which is outlined in Chap. 3. The correlation coefficient represents the linear dependency of the vectors to each other. For the attack it is assumed that a higher linear dependency between these vectors implies a more well performing model. The attacked PUF is approximated by adapting its reliability behavior of the responses of every evaluated challenge. The correlation coefficients are computed of the combinations between the one reliability vector and all hypothetical reliability vectors of all models that exist in the current generation of the algorithm.

These correlation coefficients are used as the fitness values of the corresponding model. How many models in the current generation of the algorithm exist is introduced in the next Sec. 7.2.2.

7.2.2 CMA-ES

After we showed in the previous section how to assess the fitness values of the created models, this sections states how the models are trained by the algorithm. A pseudocode representation of the CMA-ES algorithm is displayed in Alg. 3. In this section we analyze the algorithm step by step and point out the purpose of each step. In lines 1 to 3 multiple values are initialized, which are specified when used in the algorithm. Since the ES algorithm is based on the natural evolution process survival of the fittest, as termed in Sec. 3.1, the algorithm creates new models every generation.

Algorithm 3: reliability based covariance matrix adaptation (CMA) evolution strategy (ES) attack

```
Result: mean values and rates model of the last generation
 1 initialization of the number of new models per generation
 2 initialization of the values used for the CMA:
 3   means, sigma, covariance matrix
 4 while no termination criteria is fulfilled do
 5     for number of new models per generation do
 6         create new model from the values:
 7           means, covariance matrix, sigma, and random variables
 8         rate the fitness of the model by the fitness function
 9     end
10     sort models by their fitness values
11     update the values used in the CMA:
12       means, covariance matrix, sigma
13 end
```

This process of repeating generations is realized by the loop starting in line 4 and ends with fulfilling one of the termination criteria that are named later. In every generation the following steps are executed.

By the loop starting in line 5 new models are created and their fitness values are determined by the reliability based fitness function, stated in Sec. 7.2.1. The number of new created models per generation is defined by the initially set number of new models per generations in line 1. After all models are built and their fitness rated, they are sorted descending by their fitness values in line 10.

Since the ES algorithm requires a selection process of the fittest models in every generation, this is done by the CMA. The CMA adapts the values that are used for creating new models in the next generation by the values of models of the current generation in line 11 and 12. Descending sorted models of the current generation are combined with weights to rank their importance for adapting these values. The impact of a model on the values, used to build the models of the next generation, depends on its fitness value through these weights. This makes sure that the algorithm approximates fitter models.

The values that are approximated by the CMA are the means of the distributions of the values that define a model. These means in combination with random values, the covariance matrix, and the sigma value are used to create the values for the models of every generation in line 6 and 7. The covariance matrix represents the correlation of the model values to each other. The model values determine the response of the Arbiter PUF model, as outlined in Sec. 4.3.2, and thus are the values the algorithm learns. In addition, the error boundary ϵ that is stated in Sec. 7.2.1 is approximated by a value of the covariance matrix.

The sigma value is a dynamic step-size value that adjusts the added random values to build new models. The value of sigma is reduced when the learning success lowers. This means it subsides when the algorithm has correlated a well performing model to increase the likelihood of approximating a better performing model that values differ only slightly to the learned one. Supplementary, sigma sinks when the algorithm can not approximate a model, so that the algorithm terminates.

The means, the covariance matrix, and the sigma value are updated every generation in line 11 and 12. The means are updated by the ranked and weighted models of the current generation. To update the covariance matrix and the sigma value evolution paths are used. These paths represent correlations between consecutive steps of the modifications of the means. Their purpose is to reach a faster approximation and to prevent from early adaption, as mentioned in Sec. 3.2.

The generation loop from line 4 to 13 in the Alg. 3 stops with meeting one of the termination criteria. The algorithm uses the following three termination criteria:

- Maximum limit of generations reached
- Minimum limit of sigma
- Maximum fitness limit exceeded by at least one trained model

The first criteria exits the algorithm after a given number of generations. The attack ends also if the step-size sigma becomes to low and the algorithms has approximated a solution or can not approximate a solution in this run. Termination criteria three pertains when a model exceeds a given maximum fitness value. Consequently, at least one of the trained models has to be rated above this given fitness value.

After the algorithm terminated, the result are the approximated means and the rated models of the last generation. From this point on, the means or the models itself can be used to evaluate challenges of the test set to measure the performance of the trained models. If a high proportion of the challenges of the test set is evaluated correct, the reliability based CMA-ES attack will have successfully attacked an Arbiter PUF. Both the means and the models of the last generation can be used, whereas the means are expected to be better trained then the best trained model of the last generation [24].

How high this proportion has to be depends on how many correct evaluated responses are necessary to impersonate the PUF. However, this depends on the protocol that is based on the PUF and is therefore not part of this thesis.

7.2.3 Attack Combination

The reliability based CMA-ES, as outlined in the previous sections, is applied to Arbiter PUFs and Majority Arbiter PUF. To apply the attack to large XOR Arbiter PUFs and Majority XOR Arbiter PUF it has to be performed multiple times in combination with a CMA-ES attack that is based on the responses of the PUF [4].

One way to display the combined attacks is Alg. 4. For a better understanding we divide the attack in two stages, whereas line 3 to 5 contains stage one and line 7 to 9 stage two. Before the attacks start a training set and its reliability values, as clarified in Sec. 7.2.1, are created by the XOR Arbiter PUF in line 1.

Algorithm 4: combined attacks on XOR Arbiter PUFs

```
Result: mean values and rates model of the last generation
1 create training set and reliability values by attacked XOR Arbiter PUF
2 while not reached number of trained models by stage one do
3     reliability based CMA-ES attack
4 end
5 while not approximated a well performing model do
6     response based CMA-ES attack
7 end
```

In stage one the purpose of the reliability based CMA-ES attack is to approximate underlying Arbiter PUFs of the XOR Arbiter PUF one by one. This can be done, since the reliability of the challenges, evaluated by the XOR Arbiter PUF, depends "equally on each of the n employed Arbiter PUFs," as Becker mentions [4]. Accordingly, the unreliable response of one underlying Arbiter PUF results in an unreliable response of the complete PUF. Because of that, the reliability values measured of the XOR Arbiter PUF can be used to attack the underlying Arbiter PUFs with the reliability based CMA-ES attack, as described in Sec. 7.2.2.

One execution of the reliability based CMA-ES attack approximates one of the Arbiter PUFs. Multiple executions of the attack lead to different models of underlying Arbiter PUFs. As the attack is not deterministic, it can approximate models that have a response behavior similar to already approximated models. These models are eliminated by the hamming distance of their responses and the responses of already approximated models. This approach makes it possible to approximate models for different Arbiter PUFs used for a XOR Arbiter PUF. Thus, the reliability based attack is repeated by the loop in line 2 till a given number of trained models is reached.

These models, approximated by the attack in stage one and defined by their mean values, are past on to the stage two. To be able to combine all approximated models of Arbiter PUF to a XOR Arbiter PUF model, the attack of the stage two approximates all the rest of the so far not approximated models. In contrast to the attack in stage one its fitness function scores the models based on their responses of the challenges from the training set. This response based

CMA-ES attack is repeated by the loop of line 7 in Alg. 4 till a model is approximated that evaluates a high proportion of the challenges of the training set correct.

The results of the attack, similar to the normal CMA-ES attack, are the means and the approximated models of the last generation. Both the means or the models can be used to evaluate the challenges of the test set, whereas the means are expected to be better trained [24]. If they evaluate a high proportion of the test set challenges correct, the attack will be successful and a well performing model has been approximated.

As mentioned in the end of Sec. 7.2.2, the value of this proportion is not part of this thesis.

8 Stability Simulation

This section presents the results of the simulations that have been done to scrutinize the impact of MV on the ability to build large Majority XOR Arbiter PUFs. The first simulation verifies if the introduced principle of MV improves the stability of Arbiter PUF. An improvement of the stability of the Arbiter PUFs is the requirement to create large XOR Arbiter PUFs. In the second simulation the needed number of votes for MV in combination with XOR Arbiter PUFs, to reach a given stability, is studied. Finally, the stability distribution of Majority XOR Arbiter PUFs is simulated to compare its results with the stability distribution of an Arbiter PUF that is used as reference.

To approximate the stability values of PUFs and to assess the impact of MV, simulations with similar configurations are required. All simulations use a $\sigma_{\text{StageNoise}}/\sigma_{\text{Model}}$ ratio of $1/30$ and additional $\sigma_{\text{ArbiterNoise}} = 0.5$. Yet since this is a comparison, the ratio does not matter, as explained in Sec. 7.1.

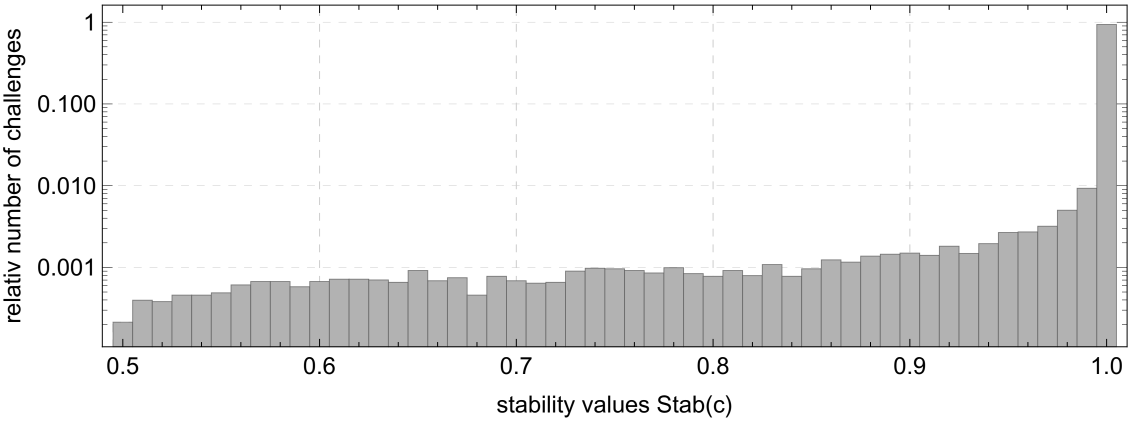

Figure 8.1: Challenge stability distribution of challenges evaluated by an Arbiter PUF using 32 stages. The graph displays buckets of challenges that are clustered by their stability values. Every challenge has been measured 100 times. The y-axis of the graph shows the number of challenges in relation to all challenges and is scaled logarithmic to figure the wide range of the values.

To be able to compare the results of simulations that including MV, reference values of Arbiter PUFs simulations without MV are needed. Consequently, Fig. 8.1 shows the stability distribution of an Arbiter PUF with $n = 32$. Every challenge has been evaluated 100 times to approximate its stability.

The very high peak for $\mathrm{Stab}(c) = 1$ states that the majority of the challenges already have a very high stability, i.e. showed the same result in all 100 evaluations. While the number of the other challenges increases for an increase of the stability, i.e. the lower the stability the fewer challenges exist. All challenges with a stability values below a given value are the unstable challenges, as described at the end of Sec. 4.1.

Since Arbiter PUFs can have a different number of stages, it could be claimed that this has impact on the stability of the PUF. Except for very low numbers of n, Fig. 8.2 displays that there is no relation and therefore no stability improvement due to the number of used stages. In this simulation for every value of n the average of the results of 10 independent Arbiter PUFs has been approximated. A challenge is here considered to be unstable if $\mathrm{Stab}(c) < 95$ %.

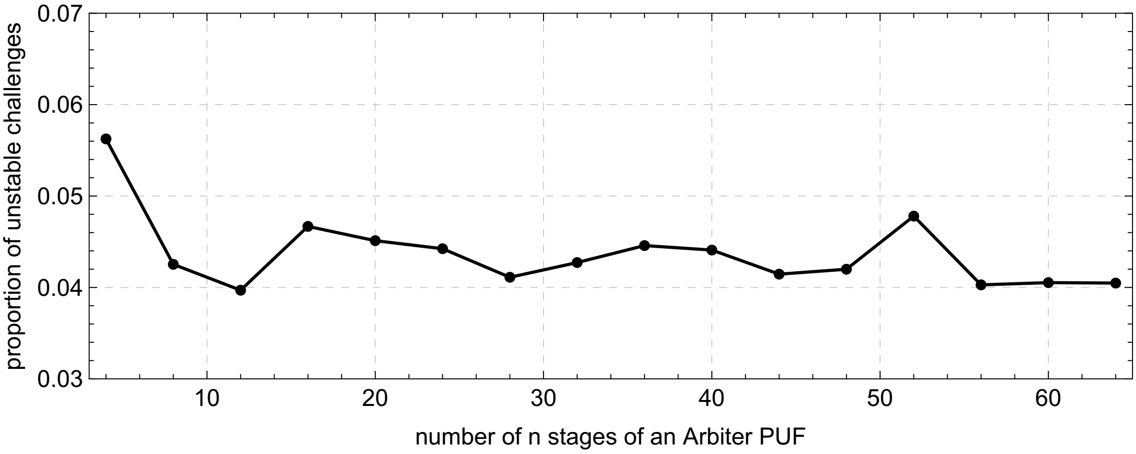

Figure 8.2: Proportion of unstable challenges of a random chosen set of challenges for Arbiter PUFs with n stages. A challenge is unstable when $\mathrm{Stab}(c) < 95$ %. The results are the average values over 10 individual PUF instances with the same challenge set.

8.1 Stability Improvement by Majority Vote

To prove the stability increase achieved by MV, Fig. 8.3 shows the decrease of unstable challenges for a growing number of votes. The Majority Arbiter PUF has size $n = 32$ and used challenges are randomly chosen. Challenges are again considered to be unstable when $\text{Stab}(c) < 95\ \%$.

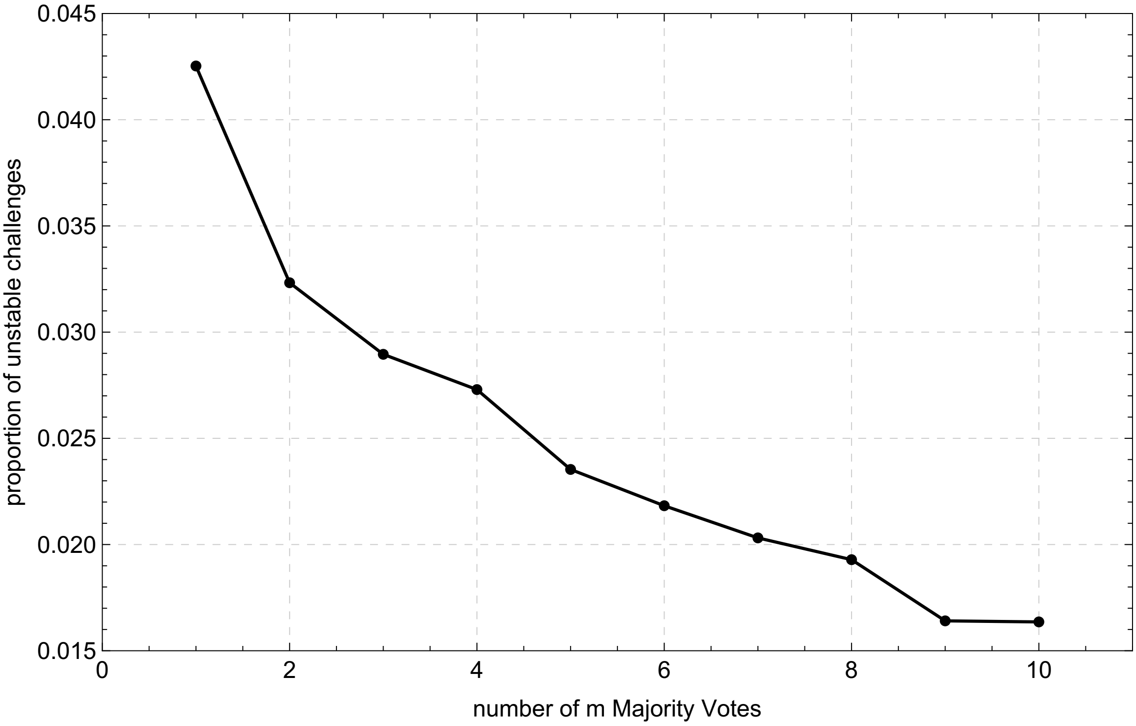

Figure 8.3: The line graph displays the decrease of the proportion of unstable challenges, which have $\text{Stab}(c) < 95\ \%$, of a random chosen set of challenges for different number of m MVs of a Majority Arbiter PUF. The Majority Arbiter PUF has size $n = 32$.

Beside the expression of declining unstable challenges the graph has a raising gradient, as Fig. 8.3 displays. This could lead to the assumption that the improving impact of MV subsides with growing m. To clarify this the logarithmic Fig. 8.4 has been created with the same values. It displays a nearly linear graph that suggest that there is no mitigation of the stability improving impact of MV with increasing m. Nevertheless, the relative stability improvement is highest with the first increase of m, as the beginning of the graph in Fig. 8.3 shows.

The Fig. 8.3 and Fig. 8.4 show that MV improves the stability of an Arbiter PUF remarkable. For more details on the challenge stability distribution Fig. 8.5 shows the challenge stability distribution of a Majority Arbiter PUF with $m = 10$

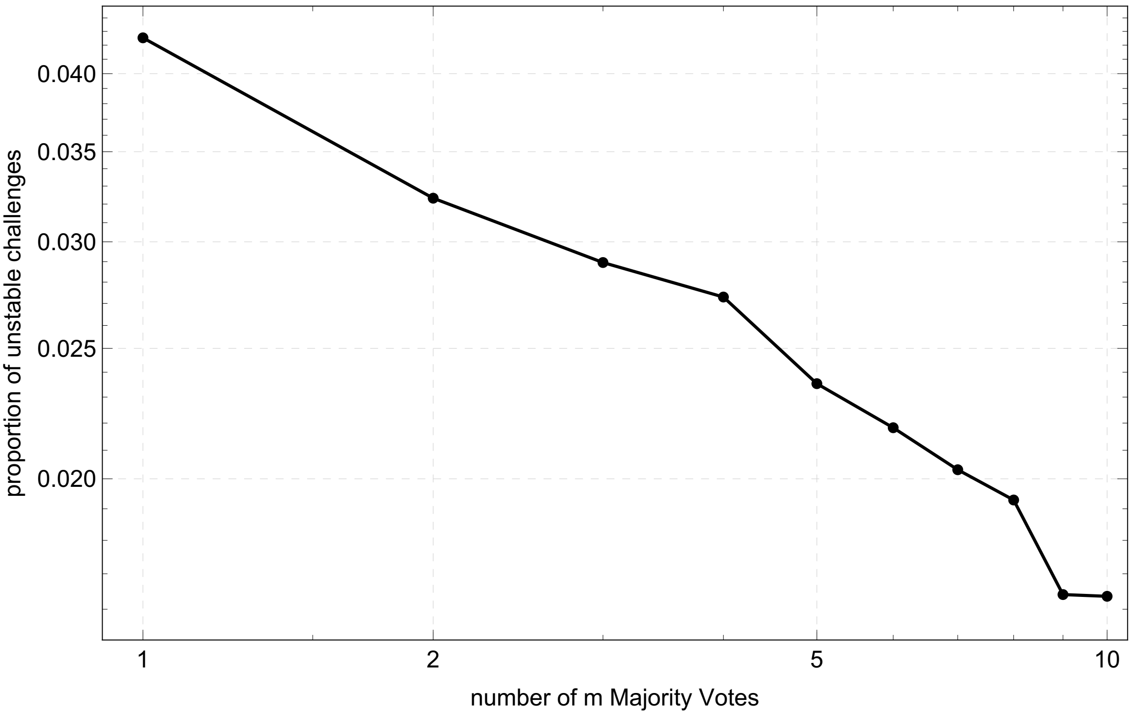

Figure 8.4: The line graph shows the decrease of the proportion of unstable challenges similar to Fig. 8.3, yet in logarithmic scale. The linear course of the graph suggests that there is no mitigation in the stability improving impact of MV with growing m used for Majority Arbiter PUFs.

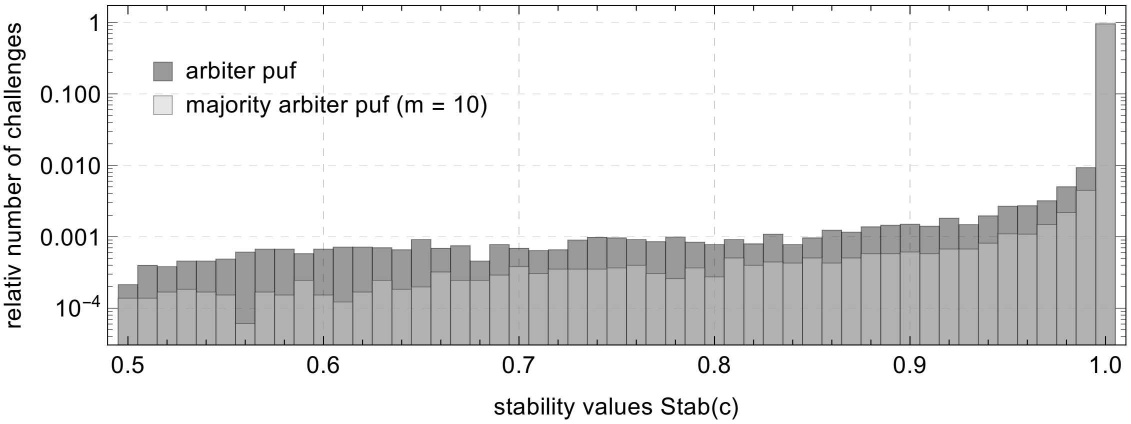

Figure 8.5: Challenge stability distribution of challenges evaluated by a Majority Arbiter PUF using 32 stages and $m = 10$ in comparison to the distribution of an Arbiter PUF, as shown in Fig. 8.1. Challenges that have been measured 100 times each are clustered by their stability. The y-axis displays the number of challenges relative to all challenges and is scaled logarithmic to figure the wide range of the values.

MVs in comparison to the challenge stability distribution of an Arbiter PUF, as shown in the beginning of this chapter. In the comparison of the figures the reduced number of challenges with lowered stability can be seen. This stability increase of Majority Arbiter PUFs can be used to build large Majority XOR Arbiter PUFs, which are examined in Sec. 8.2.

8.2 Majority Vote Increase for Majority XOR Arbiter PUFs

As only XOR Arbiter PUFs that are large in k can not be modeled successfully by known attacks, it is the purpose to be able to create large XOR Arbiter PUFs, which have to be stable to produce a reliable output. To achieve this through MV, the number of m votes has to be increased with a growth in k, as shown in Sec. 6.2.

Yet the growth rate of m must not be too high otherwise the time to evaluate a challenge would grow too large with large XOR Arbiter PUFs as well. To show that this high stability can be reached using a polynomial number of m MVs, multiple Majority XOR Arbiter PUFs are simulated. These Majority XOR Arbiter PUFs consists of different numbers of k parallel Majority Arbiter PUFs.

The simulation approximates the number of m MVs to reach $\Pr[\text{Stab}(c) \geq 95\,\%] \geq 95\,\%$ for different numbers of k by binary search. Therefore, it measures the stability values of every challenge evaluated by the underlying Majority Arbiter PUFs through multiple evaluations, calculates the stability values for the Majority XOR Arbiter PUF, and changes the number of m votes till the stability is reached.

Fig. 8.6 shows how many votes are needed to reach a likelihood of 95 % that uniformly and randomly chosen challenges have a stability of at least 95 %, for different values of k. These are upper bounds, as the XOR operation increases the stability of a XOR Arbiter PUF marginal, because any even number of incorrect Arbiter PUF responses will still yield the desired output of the XOR Arbiter PUF. This behavior is not included in the simulation, since the stability values of the challenges are approximated for all Arbiter PUFs independently.

The polynomial increasing number of votes needed to fulfill the stability requirements for a useful PUF implementation displayed in the Fig. 8.6 proves that XOR Arbiter PUFs can be built arbitrary large using MV.

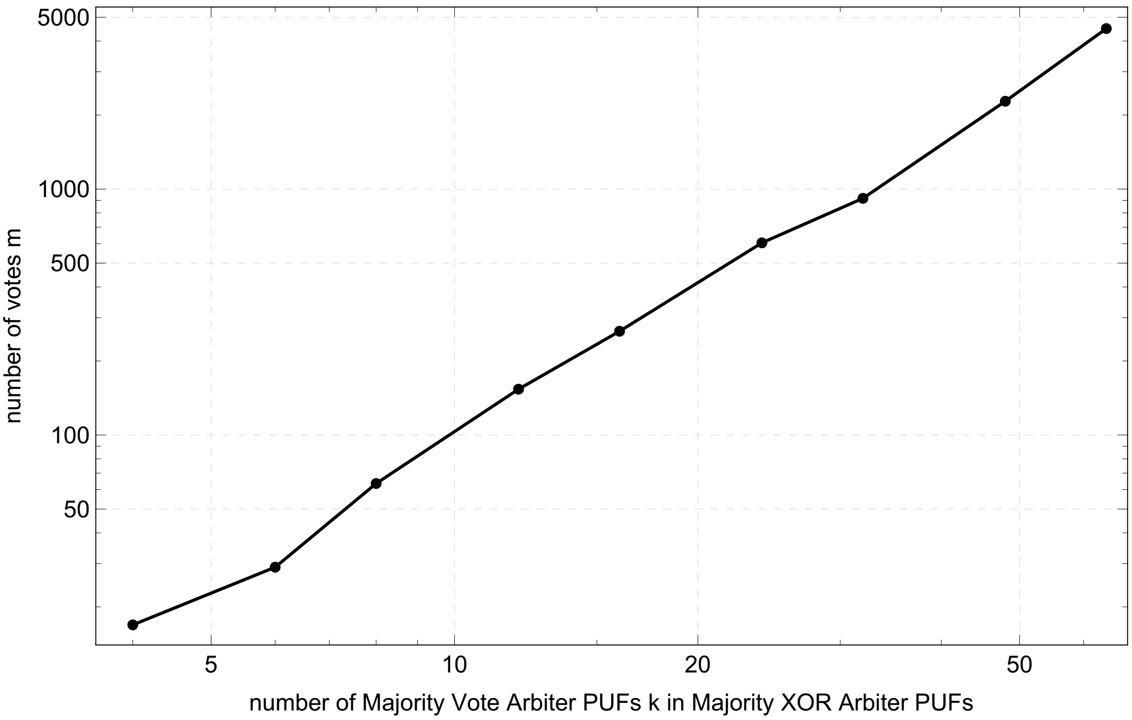

Figure 8.6: Minimum number of votes needed to reach $\Pr[\text{Stab}(c) \geq 95\ \%] \geq 95\ \%$ for uniformly and randomly chosen challenges c for different numbers of k Arbiter PUFs. The simulations use $n = 16$, but the results are independent of n, as shown in Fig. 8.2. The linear graph shown in a log-log-graph confirms the polynomial growing number m of MVs of the degree 1. All results are the average values over 10 individual PUF instances with independent challenge sets.

8.3 Stability Distribution of Large Majority XOR Arbiter PUFs

To get a more precise exposition of the stability distribution of Majority XOR Arbiter PUFs with large size k, the stability value for every challenge of a subset of all challenges is simulated. The results are then compared with the stability distribution of the Arbiter PUF.

Only a subset of challenges is used due to the needed simulation computation time. Nevertheless, this random chosen subset represents the distribution of all challenges. All challenge of the subset are evaluated 100 times to compute their stability values. The example uses 32 Majority Arbiter PUFs with 32 stages each. The number of MVs needed to reach a stability of 95 % with a likelihood of 95 % is chosen from the results of Sec. 8.2.

The challenge stability distribution of a Majority XOR Arbiter PUF, shown in Fig. 8.7, was achieved by using $m = 600$ votes and is roughly equivalent to the distribution of an Arbiter PUF displayed in the Fig. 8.1 and here for comparison.

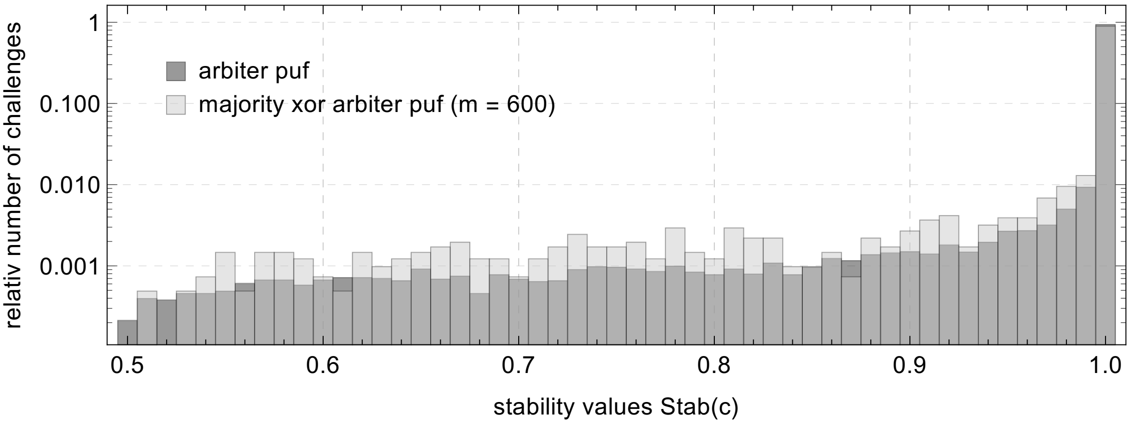

Figure 8.7: The histogram shows the relative frequency of the challenge stability values clustered into buckets of a Majority XOR Arbiter PUF with size $k = 32$ and 32 stages each in comparison to the challenge stability distribution of an Arbiter PUF, as shown in Fig. 8.1. MV uses $m = 600$ to reach a stability of 95 % with likelihood 95 % for randomly chosen challenges. The distribution and the amount of unstable challenges of the Majority XOR Arbiter PUF resembles the distribution of an Arbiter PUF.

This shows that even large Majority XOR Arbiter PUFs can become stable through a feasible number of votes and even exceed stability of a basic Arbiter PUF with a larger number of MV.

9 Attack Simulation

The combined reliability based CMA-ES is the only attack, which is claimed to have a linear increasing complexity for successful attacks on XOR Arbiter PUFs of arbitrary size, as indicated in Sec. 5.3 [4]. Successfully attacked here means to create a model that can predict 80 % of the challenges from a test set, which contains randomly chosen challenges, correct. The implementation details of the CMA-ES attack and the Arbiter PUF, which is basis for the simulations of the Majority Arbiter PUF, is explained in Chap. 7.

This section presents the results of the simulations to verify the success of the CMA-ES attack on large XOR Arbiter PUFs. First, the dependency between the number of vote used for MV and the number of collected unreliable challenges is examined, as the attack is based on the unreliability of challenges. The normal reliability based CMA-ES attack is part of the combined attack on Majority XOR Arbiter PUFs and attacks the underlying Majority Arbiter PUFs. Hence, in the simulation the success of this normal reliability based CMA-ES attack on Majority Arbiter PUFs in comparison to Arbiter PUFs is evaluated, since they are used to build the Majority XOR Arbiter PUF. Finally, the CMA-ES attack is applied to XOR Arbiter PUFs and Majority XOR Arbiter PUFs of different size to study whether MV can prevent the attack through enabling large Majority XOR Arbiter PUFs.

9.1 Detection of Unreliable Challenges

For the reliability based CMA-ES attack on Arbiter PUFs it is crucial to find challenges, which evaluate differently at multiple evaluations, as termed in Sec. 7.2.1. To find these challenges and their reliability h every challenge c of a set will be evaluated j times. Consequently, the reliability h will be computed for every challenge by Eq. (7.1). Note that the notions of stability and reliability are slightly different for the CMA-ES attack, as defined in Sec. 7.2.1.

When using MV to increase the stability of the Arbiter PUF, finding challenges with lower reliability becomes more difficult. Accordingly, Fig. 9.1 shows the relation between the increase of votes m and the decrease of found challenges with lowered reliability h. In this case, a lowered reliability is already reached if a challenge c evaluates one or multiple times different to the rest of the j evaluations and is thus called unreliable. There are four line graphs for the different numbers of evaluations j.

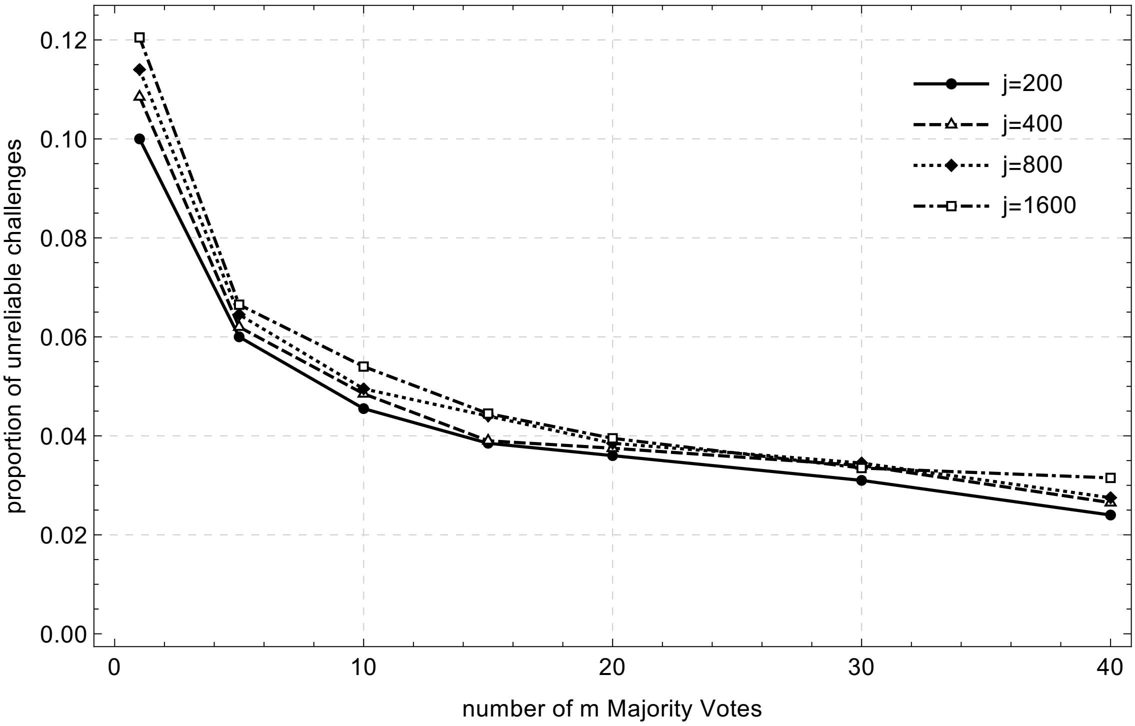

Figure 9.1: Proportion of unreliable challenges for different values of m MVs. The different line graphs represent the number of evaluations j in 200, 400, 800, 1600. The slight distance between the graphs in the beginning shows the very little increase of found unreliable challenges with increasing number of evaluations j and how this gap resolves with introducing of MV.

The raising gradient of all four graphs could lead to the same assumption of the subsiding impact of MV with rising m, as mentioned in Sec. 8.1. However, the linear course of the graph, showed by the logarithmic Fig. 9.2, suggest that this is not the case and supports the result of Sec. 8.1.

The slight difference between the graphs in both figures shows that increasing the number of evaluation j does not necessarily provide more challenges with lowered reliability h. Without MV there is a little increase of found challenges with lowered reliability h. Yet with applying MV this increase disappears with growing m.

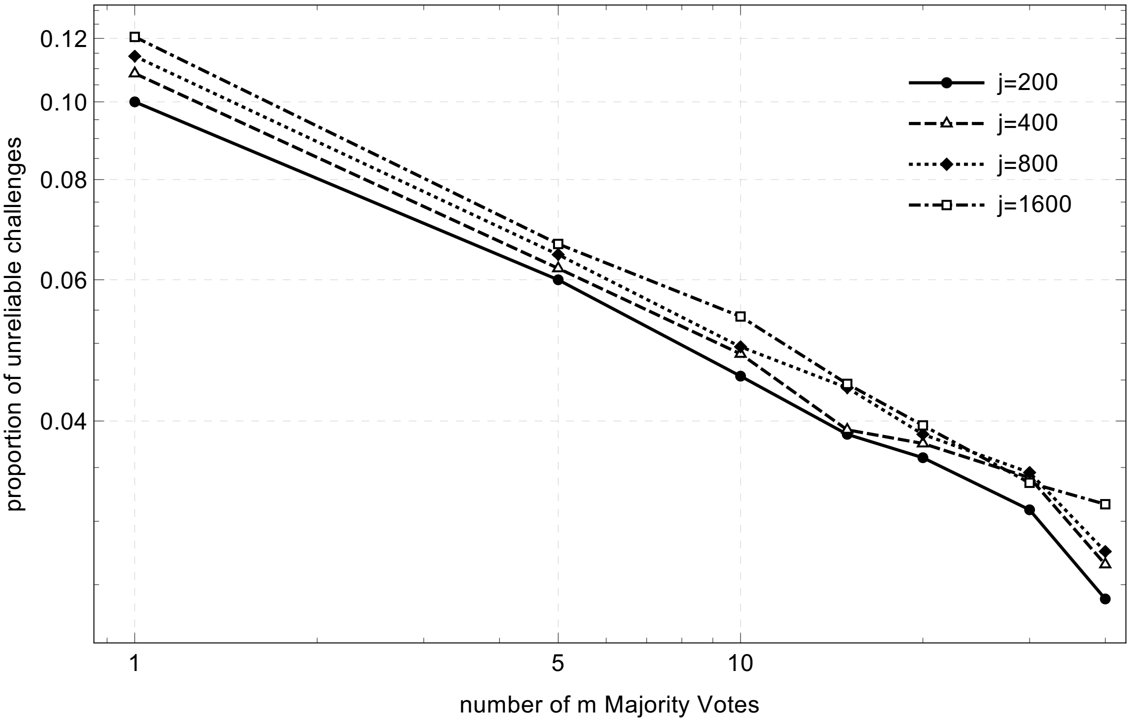

Figure 9.2: The line graphs show the proportions of unreliable challenges for different values of m MVs similar to Fig. 9.1, yet in logarithmic scale. The linear course of the graphs suggest that there is no decline of the stability improving impact of MV with increasing m used for Majority Arbiter PUFs.

The results presented in this section verify that the number of unreliable challenges decreases with an increase of used MV votes. Despite the use of MV, unreliable challenges are occurring.

9.2 Arbiter PUFs vs. Majority Arbiter PUFs

This section describes the results of the simulation used to scrutinize the success of the reliability based CMA-ES attacks, presented in Sec. 7.2.2, on Majority Arbiter PUFs in comparison to Arbiter PUFs. Through the comparison with the success of the attack on Arbiter PUF the impact of MV can be evaluated.

The Arbiter PUF used for the simulation has $n = 32$ stages. For the attack a training set of 10000 CRPs and a number of evaluation times of $j = 100$ were used. Since the CMA-ES attack not always approximates to a model, which performs well on the challenges of the test set, it has to be restarted. The number of needed starts of the attack till a desired model has been approximated is called number of attack executions.

A model produced by the attack is defined to perform well if it evaluates 80 % of the 10000 CRPs of the test set correct. Both test set and training set are independent randomly chosen sets of challenges c. The minimum step-size sigma, used in the attack as termination criteria, explained in Sec. 7.2.2, is set to the value 0.2. Reaching this limit stops the attack regardless if a well performing model is approximated or not.

For a growing number of MV m there is an increase of needed executions of the attack to approximate to a model, which performs well on the test set, as shown in Fig. 9.3. The growing gradient of the graph shows the growing exponential increase of needed attack executions to gain well performing models. From $m = 80$ MVs on the gradient of the graph is greater than 1. Hence, the slope of the number of attack executions is greater than the polynomial degree 1.

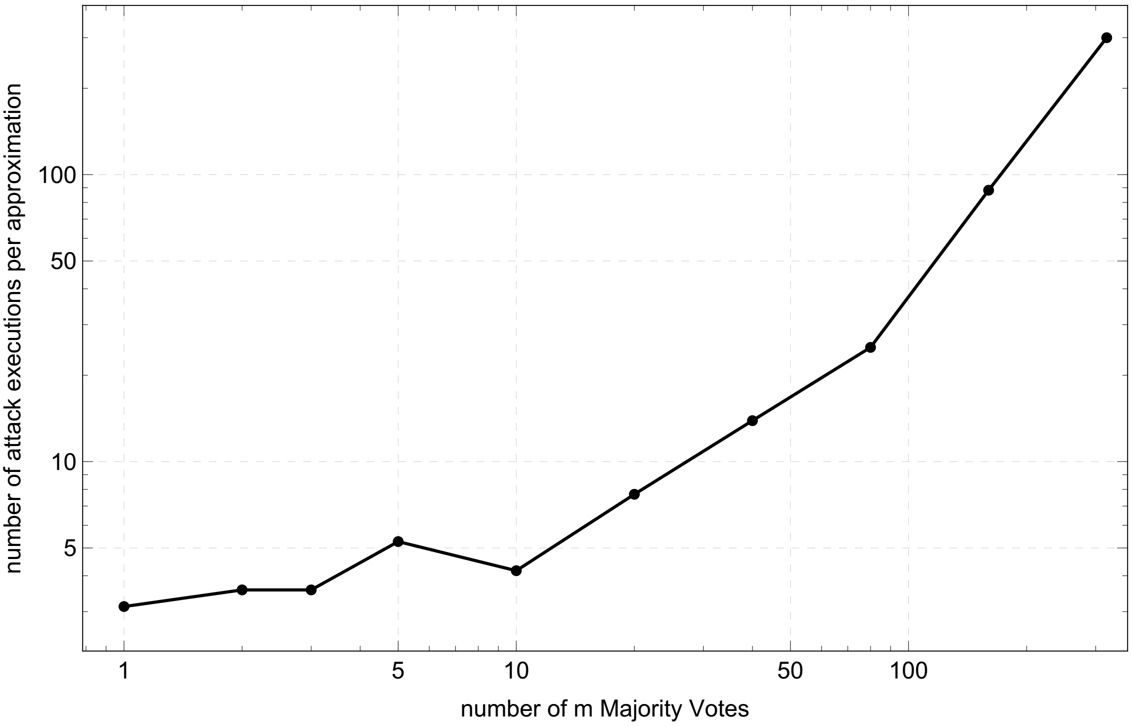

Figure 9.3: Number of attack executions per approximation to a well performing model. A well performing model evaluates 80 % of the 10000 test set CRPs correct. The rising gradient of the graph shows the growing exponential increase of the number of attack executions.

Apart from the increasing number of attack executions the performance of the computed models decreases with the growth of m, as shown in Fig. 9.4. The graph displays how the proportion of correct evaluated challenges c of the test set lowers linear with a linear rise of the number of MV m. This indirect correlation implies that with growing number of MV m the proportion of the correct evaluated challenges c of the test set decreases for well performing models built by the CMA-ES attack. The values of both Fig. 9.3 and Fig. 9.4 are average values of minimum 10 well performing models.

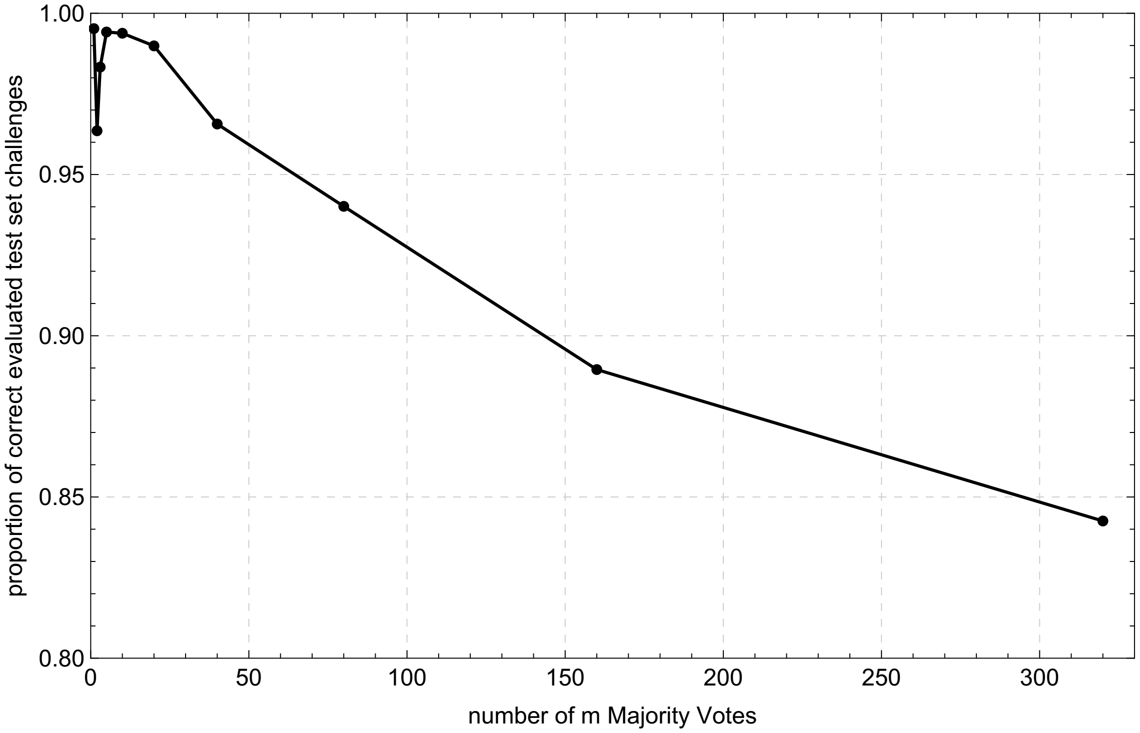

Figure 9.4: Proportion of correct evaluated test set CRPs of a well performing model built by a CMA-ES attack. The linear decrease of the graph shows the decrease of the proportion of correct evaluated challenges c by a well performing model. The values used are average values of minimum 10 well performing models.

The results of this section show that the required number of CMA-ES attack executions increases faster than the growing number of votes m from a certain point on. Applying a large number of votes can be used to minimize the likelihood of a successful attack. MV improves the ability of Arbiter PUFs to resist reliability based CMA-ES attacks.

9.3 XOR Arbiter PUFs vs. Majority XOR Arbiter PUFs

This section presents the results of the simulation to verify the success of the combined CMA-ES attacks, as introduced in Sec. 7.2.3, on Majority XOR Arbiter PUFs in comparison to XOR Arbiter PUFs. Through the comparison with the success of the attack on XOR Arbiter PUF the impact of MV can be evaluated. First, the results of the attack on XOR Arbiter PUFs are compared with the results of the attack on Majority XOR Arbiter PUFs. Subsequently, limits of the attack on Majority XOR Arbiter PUFs are outlined. Finally, the modifications of the preconditions for the attack and the according results are shown.

To gain comparable results the simulation uses the same preconditions for the attacks on XOR Arbiter PUFs and Majority XOR Arbiter PUFs. All PUFs in this simulation have $n = 32$ stages. The number of challenges in the training set increases linear with the growth of the number of k Arbiter PUFs or k Majority Arbiter PUFs. The simulations start with 5000 training set challenges for $k = 2$ and doubles the number of training set challenges for the double number of k. All training set challenges are individually and randomly chosen for all attacks.

A test set is used to evaluate the performance of the learned model. The attack is successful when the learned model evaluates 80 % of the test set challenges correct and thus is defined to perform well. The test set contains 2000 individually, randomly chosen challenges for all measurements, since only the complexity of the attack grows for similar results. A number of 5000 training set challenges in combination with 2000 test set challenges were chosen for the reason that attacks on XOR Arbiter PUFs with $k = 2$ and $k = 4$ with accordingly 10000 training set challenges were able to build well performing models.

Fig. 9.5 shows the proportion of correct evaluated test set challenges by well performing models produced by the attacks for $2 \leq k \leq 14$. Every value is the average of 10 successfully learned models. Since the attack is not deterministic, only attack executions that approximate well performing models are counted. The number of required attack executions for a successful attack are similar for all attacks on XOR Arbiter PUFs. In the case of Majority XOR Arbiter PUFs the number of attack executions needed to obtain a well performing model rose between $k = 3$ and $k = 4$ by the factor 5. Yet for $k = 4$ and $k = 6$ similar numbers of attack executions per well performing model are necessary.

The numbers of MV votes m used for the Majority XOR Arbiter PUFs in Fig. 9.5 are determined by the results of the simulations in Sec. 8.2. These simulations approximated the required numbers of votes to gain stable large Majority XOR

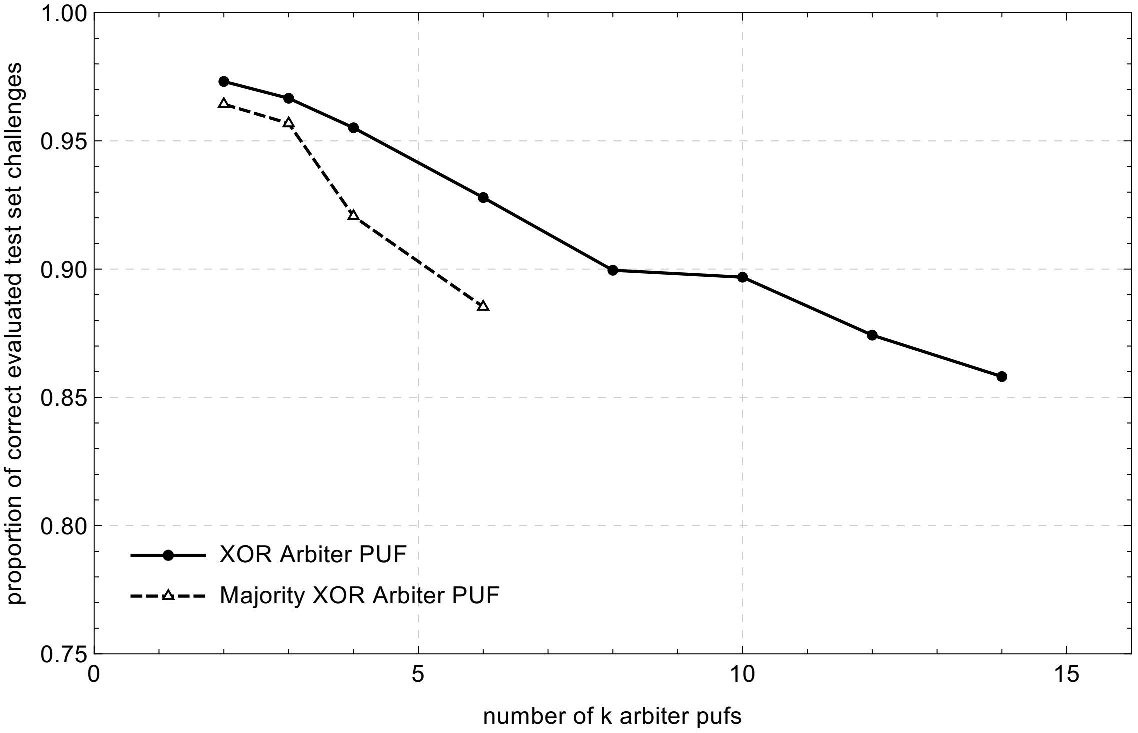

Figure 9.5: Proportion of correct evaluated test set challenges of XOR Arbiter PUF models and Majority XOR Arbiter PUF models. All values are average values of 10 independent attacks. The number of used training set challenges and votes m is chosen to accord with the Tab. 9.1. The test set contains for all attacks 2000 challenges.

Arbiter PUFs, which means to reach $\Pr[\text{Stab}(c) \geq 95\ \%] \geq 95\ \%$, for different k. In Tab. 9.1 the used values m for different values of k and the number of training set challenges that are the same for XOR Arbiter PUFs and Majority XOR Arbiter PUFs are displayed.

number of k	2	3	4	6	8	10	12	14
number of training set challenges ($\cdot 10^3$)	5	7.5	10	15	20	25	30	35
number of votes m	5	9	17	29	64	100	150	200

Table 9.1: Number of used training set challenges for the attack on XOR Arbiter PUFs and Majority XOR Arbiter PUFs. Number of MV votes used for the according Majority XOR Arbiter PUFs determined by the simulation in Sec. 8.2.

The Fig. 9.5 states the decrease of the performance of models, which are approximated of XOR Arbiter PUFs. With this decrease in performance a continuous trend would already lead to not well performing models learned by an attack on large XOR Arbiter PUFs. The attack would not be successful for large XOR Arbiter PUFs, if it was possible to build them stable. If there was no impact of

MV on the attack success, the models, approximated by the attack on Majority XOR Arbiter PUFs, would reach the same performance. For the case that the attack benefits of MV, the performance values of the attack on XOR Arbiter PUFs would be exceeded. However, the line graph for well performing models of the attack on Majority XOR Arbiter PUFs stops after $k = 6$. The attack was not able to learn well performing models of Majority XOR Arbiter PUFs that have a $k \geq 8$ and uses m votes and the number of challenges, as defined in Tab. 9.1.

Nevertheless, the attack could be successful and train models, which fulfill a certain performance, with a polynomial increase in the number of used training set challenges. To study the success of the attack on Majority XOR Arbiter PUFs, we approximate the required training set CRPs for successful attacks that lead to models with a given continuous performance by binary search. In Fig. 9.6 a comparison between the required training set challenges of the attack on XOR Arbiter PUFs and Majority XOR Arbiter PUFs is shown. The raising gradient of the graph states the exponential increase in demanded CRPs for a successful attack on Majority XOR Arbiter PUFs that use the according number of votes m, as display in Tab. 9.1.

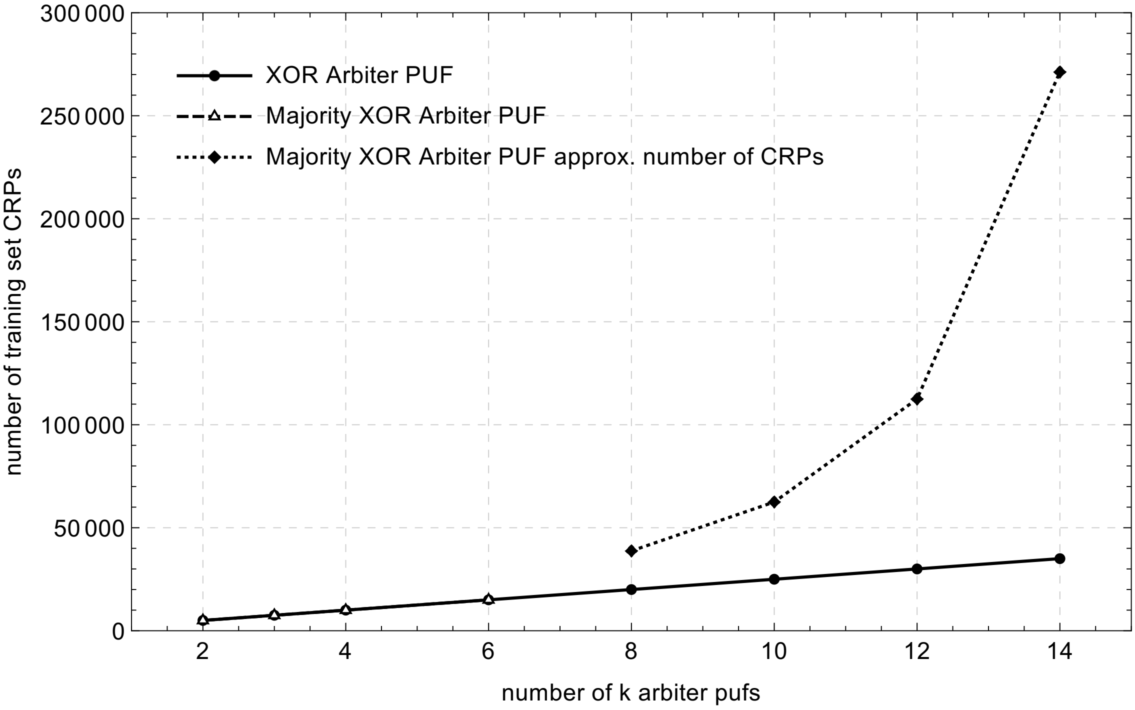

Figure 9.6: Number of used training set challenges to attack different PUFs. The necessary number of training set challenges for well performing model with a continuous performance are approximated by binary search.

To scrutinize the approximated required number of training set CRPs, 10 successful attacks for every *k* are performed. Then the average proportion of correct evaluated test set challenges by the well performing models are compared with the results of the attacks on XOR Arbiter PUFs in Fig. 9.7. The continuous high performance of the models, which are learned with the approximated number of training set challenges, shows that the large increase of required challenge in Fig. 9.6 are needed to obtain well performing models of the attacks. This is important, since otherwise with a growing number *k* the performance of the models would subside and the attack would not be successful.

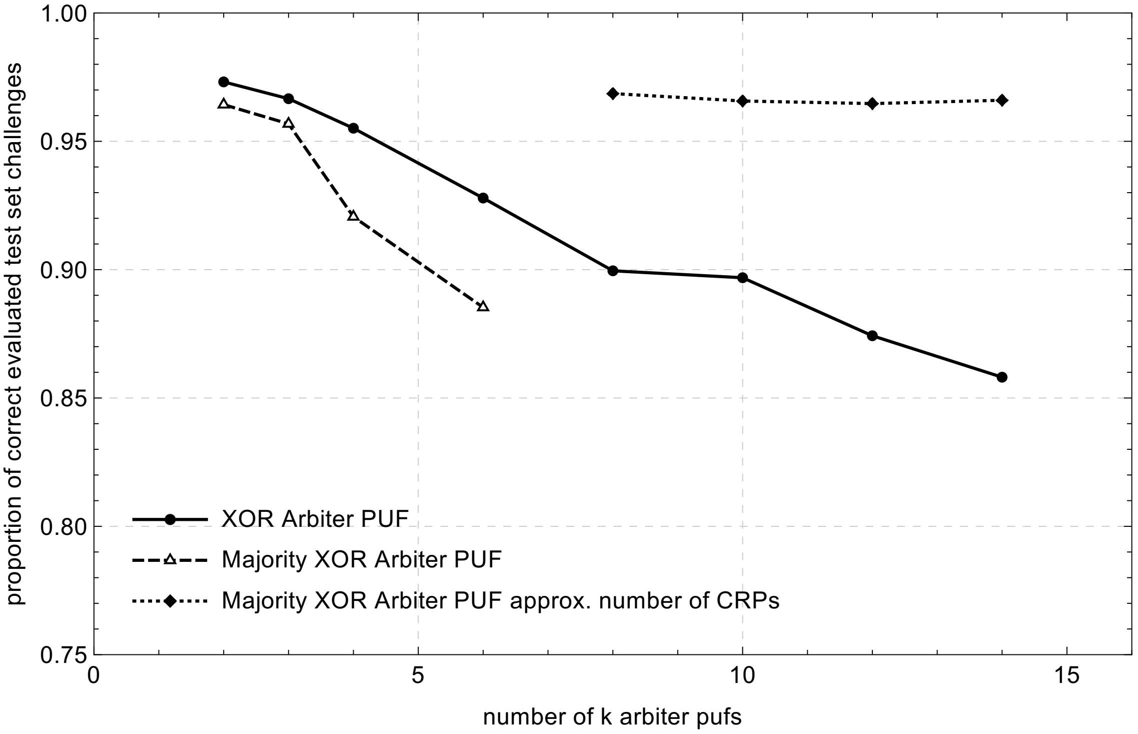

Figure 9.7: Proportion of correct evaluated test set challenges of well performing models trained with an approximated number of training set challenges. To measure the performance of the model a test set with 2000 CRPs is used and the results are compared with the results of the attack simulations of Fig. 9.5. The number of used training set challenges is displayed by Fig. 9.6.

The exponential raise of the required number of training set challenges for successful attacks confirms the impact of MV on the combined CMA-ES attack. The complexity of the attack grows exponentially, since the number of needed challenges to train the model increases exponentially for Majority XOR Arbiter PUFs with a linear growth of the number of Majority Arbiter PUFs.

The claim, which states that the complexity of the combined CMA-ES attack grows linear for XOR Arbiter PUFs with linear growth in k, is not directly refuted. However, the performance of the learned model subsides and thus the attack will not be able to learn models with a certain performance for a large given number of k.

Nevertheless, Chap. 8 outlined that it is possible to build large XOR Arbiter PUFs with the principle of MV called Majority XOR Arbiter PUFs. In Sec. 5.3 we state that only the combined CMA-ES attack is relevant to large XOR Arbiter PUFs and thus Majority XOR Arbiter PUFs. As a result, large Majority XOR Arbiter PUFs can not be successfully modeled by the combined CMA-ES attack or any other currently known attack without exponentially growing attack complexity.

10 Conclusion

10.1 Future Work

The aim of this work was to verify the stability improvement of Majority Arbiter PUFs, a modified version on Arbiter PUFs, and the impact on related attacks. All parts to achieve this have been outlined, but some have not been studied in greater detail. Depending on the aim, different parts can be considered more extensively.

Despite the clear principles of Arbiter PUFs, a way to create them as circuit has only been summarized. For real physical implementations the required electronic components and their interplay has to be examined. Possibilities to overcome the problems that FPGA implementations of Arbiter PUFs face is be an issue of research [32, 33, 36, 43]. Apart from modified versions of Arbiter PUFs, other types of PUFs, e.g. controlled PUFs, can be interesting alternatives to study in the field of physical implementations [21].

This work focuses on a specific class of attacks on PUFs. The field of ML attacks received more attention in the last three years [22]. For this reason and in combination with a continuous raising available processing power, more ML attack possibilities that are worth examining can occur in the future. Another field of study are invasive attacks that can be applied to real PUF implementations. The success of those attacks and how PUFs can be protected are still crucial topics of research for using PUFs in the real world.

The security of all PUFs is based on their intrinsic information, which they contain. Yet the entropy of this information does not serve as attribute to compare their security quality [57]. In addition, PUFs are not clear defined and due to this a question of research can be to develop a method to analyze the security characteristics of PUFs that yields to a comparable result [4].

10.2 Final Comments

To conclude this thesis, we would like to point out the main insights that have been disclosed. The principle of MV applied to Arbiter PUFs and XOR Arbiter PUFs has been introduced to improve their stability and resistance against successful attacks. The idea behind MV is to evaluate the same PUF multiple times, where the most occurring response the final response is.

By simulations, it was verified that with the concept of MV it becomes possible to build large stable Majority XOR Arbiter PUFs. All non-invasive attacks on XOR Arbiter PUFs, known at the time of writing, can thus be prevented by increasing their complexity exponentially through a linear growth of the number of used Majority Arbiter PUFs. As a consequence, large Majority XOR Arbiter PUFs are resistant to all known machine learning attacks.

This gives the opportunity to use them as physical implementation of one-way hash functions or secret key memory to keep a key secret. Many goals of the information security base on the secrecy of the key. Hence, preventing machine learning attacks creates a foundation to met these goals and to keep connections, data, and other things that are worth protecting secure.

However, it is certain is that there exist no absolute secure system. In addition, capabilities of attackers raise as well as allegedly secure systems become vulnerable through e.g. new attacks that may show up. On the contrary, new security proofs could reveal the security of systems. Yet physical attacks, as an alternative to machine learning attacks, were possible by the time of writing and often not considered by the design of systems [70, 71]. As an example, the design of the studied Arbiter PUF includes no additional protection against invasive attacks [69]. With that said there is still a lot to improve till most of the vulnerabilities can be controlled. PUFs are one way to help us reach a greater level of security.

Bibliography

[1] C. Adams and S. Tavares, "The use of bent sequences to achieve higher-order strict avalanche criterion in s-box design", Dept. of Elec. Eng., Queen's University, Kingston, Ontario, Canada, Tech. Rep. TR 90-013, 1990.

[2] J. Aldrich, "R.A. fisher and the making of maximum likelihood 1912-1922", *Statist. Sci.*, vol. 12, no. 3, pp. 162–176, Sep. 1997.

[3] F. Armknecht, R. Maes, A. R. Sadeghi, F. X. Standaert, and C. Wachsmann, "A formal foundation for the security features of physical functions", *Proceedings - IEEE Symposium on Security and Privacy*, pp. 397–412, 2011.

[4] G. T. Becker, "The gap between promise and reality: On the insecurity of XOR arbiter PUFs", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9293, pp. 535–555, 2015.

[5] G. T. Becker and R. Kumar, "Active and passive side-channel attacks on delay based PUF designs", *IACR Cryptology ePrint Archive*, pp. 1–14, 2014.

[6] G. T. Becker and J. Tobisch, "On the scaling of machine learning attacks on PUFs with application to noise bifurcation", *Radio Frequency Identification: Security and Privacy Issues*, vol. 6370, pp. 17–31, 2010.

[7] D. A. Belsley, E. Kuh, and R. E. Welsch, *Regression diagnostics : Identifying influential data and sources of collinearity*. Wiley-Interscience, 1980.

[8] A. Blum, A. Frieze, R. Kannan, and S. Vempala, "A polynomial-time algorithm for learning noisy linear threshold functions", *Proceedings of 37th Conference on Foundations of Computer Science*, pp. 330–338, 1996.

[9] J. Brosow and E. Furugard, *Method and a system for verifying authenticity safe against forgery*, US Patent 4,218,674, 1980.

[10] Z. Cherif Jouini, J. L. Danger, and L. Bossuet, "Performance evaluation of physically unclonable function by delay statistics", *2011 IEEE 9th International New Circuits and Systems Conference, NEWCAS 2011*, pp. 482–485, 2011.

[11] A. Chotard, A. Auger, and N. Hansen, "Cumulative step-size adaptation on linear functions", *International Conference on Parallel Problem Solving from Nature*, pp. 72–81, 2012.

[12] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines: And other kernel-based learning methods*. Cambridge University Press, 2000.

[13] G. Csaba, X. Ju, Z. Ma, Q. Chen, W. Porod, J. Schmidhuber, U. Schlichtmann, P. Lugli, and U. Rührmair, "Application of mismatched cellular nonlinear networks for physical cryptography", *International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*, pp. 1–6, 2010.

[14] J. Delvaux and I. Verbauwhede, "Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise", *Proceedings of the 2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013*, pp. 137–142, 2013.

[15] H. Feistel, "Cryptography and computer privacy", *Scientific american*, vol. 228, pp. 15–23, 1973.

[16] R. Forrié, "The strict avalanche criterion: Spectral properties of boolean functions and an extended definition", in *Advances in Cryptology — CRYPTO' 88: Proceedings*, S. Goldwasser, Ed. New York, NY: Springer New York, 1990, pp. 450–468.

[17] F. Ganji, S. Tajik, and J.-P. Seifert, "PAC learning of arbiter PUFs", *Journal of Cryptographic Engineering*, vol. 6, no. 3, pp. 249–258, 2016.

[18] F. Ganji, S. Tajik, and J.-P. Seifert, "Why attackers win: On the learnability of XOR arbiter PUFs", in *Trust and Trustworthy Computing: 8th International Conference, TRUST 2015, Heraklion, Greece, August 24-26, 2015, Proceedings*, M. Conti, M. Schunter, and I. Askoxylakis, Eds. Cham: Springer International Publishing, 2015, pp. 22–39.

[19] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions", *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS2002*, p. 148, 2002.

[20] B. Gassend, D. Lim, D. Clarke, M. Van Dijk, and S. Devadas, "Identification and authentication of integrated circuits", *Concurrency Computation Practice and Experience*, vol. 16, no. 11, pp. 1077–1098, 2004.

[21] B. Gassend, M. Van Dijk, D. Clarke, and S. Devadas, "Controlled physical random functions", *Security with Noisy Data: On Private Biometrics, Secure Key Storage and Anti-Counterfeiting*, pp. 235–253, 2007.

[22] Google. (2017). Machine learning - google trends, [Online]. Available: https://www.google.com/trends/explore?q=machine%20learning.

[23] N. Hansen, "The CMA evolution strategy: A comparing review", *Towards a new evolutionary computation*, pp. 75–102, 2006.

[24] N. Hansen. (2011). The CMA evolution strategy: A tutorial, [Online]. Available: http://www.lri.fr/~hansen/cmatutorial110628.pdf.

[25] C. Helfmeier, C. Boit, D. Nedospasov, S. Tajik, and J.-P. Seifert, "Physical vulnerabilities of physically unclonable functions", *Design, Automation and Test in Europe Conference and Exhibition*, pp. 1–4, 2014.

[26] R. Helinski, D. Acharyya, and J. Plusquellic, "A physical unclonable function defined using power distribution system equivalent resistance variations", *ACM/IEEE Design Automation Conference*, pp. 676–681, 2009.

[27] G. Hospodar, R. Maes, and I. Verbauwhede, "Machine learning attacks on 65nm arbiter PUFs: Accurate modeling poses strict bounds on usability", *WIFS 2012 - Proceedings of the 2012 IEEE International Workshop on Information Forensics and Security*, pp. 37–42, 2012.

[28] ISO, "Information technology – security techniques – information security management systems – overview and vocabulary", International Organization for Standardization, Standard 27000:2016, 2016.

[29] J. Lee, D. L. D. Lim, B. Gassend, G. Suh, M. V. Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications", *Proceedings of 2004 Symposium on VLSI Circuits.*, pp. 176–179, 2004.

[30] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. v. Dijk, and S. Devadas, "Extracting keys from integrated circuits", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 10, pp. 1200–1205, 2005.

[31] L. Lin, D. Holcomb, D. K. Krishnappa, P. Shabadi, and W. Burleson, "Low-power sub-threshold design of secure physical unclonable functions", *International Symposium on Low-Power Electronics and Design (ISLPED)*, pp. 43–48, 2010.

[32] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "A new arbiter PUF for enhancing unpredictability on FPGA", *Scientific World Journal*, vol. 2015, 2015.

[33] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "Implementation of double arbiter PUF and its performance evaluation on FPGA", *20th Asia and South Pacific Design Automation Conference, ASP-DAC 2015*, pp. 6–7, 2015.

[34] R. Maes, V. Rozic, I. Verbauwhede, P. Koeberl, E. van der Sluis, and V. van der Leest, "Experimental evaluation of physically unclonable functions in 65 nm CMOS", *2012 Proceedings of the ESSCIRC (ESSCIRC)*, pp. 486–489, Sep. 2012.

[35] A. Mahmoud and U. Rührmair, "Combined modeling and side channel attacks on strong PUFs", *EPrint Archive*, pp. 1–9, 2013.

[36] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines", *2010 IEEE International Workshop on Information Forensics and Security, WIFS 2010*, 2010.

[37] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUFs", *2008 IEEE/ACM International Conference on Computer-Aided Design*, vol. 77005, pp. 670–673, 2008.

[38] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for design and implementation of secure reconfigurable PUFs", *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 1, pp. 1–33, 2009.

[39] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing techniques for hardware security", *Proceedings - International Test Conference*, pp. 1–10, 2008.

[40] M. M. Mano, C. R. Kime, T. Martin, *et al.*, *Logic and computer design fundamentals*. Prentice Hall, 2008, vol. 3.

[41] M. Minsky and S. Papert, *Perceptrons: An introduction to computation geometry*. MIT press, 1969, p. 258.

[42] T. M. Mitchell, *Machine learning*. McGraw-Hill, Inc., 1997.

[43] S. Morozov, A. Maiti, and P. Schaumont, "An analysis of delay based PUF implementations on FPGA", in *Reconfigurable Computing: Architectures, Tools and Applications: 6th International Symposium, ARC 2010, Bangkok, Thailand, March 17-19, 2010. Proceedings*, P. Sirisuk, F. Morgan, T. El-Ghazawi, and H. Amano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 382–387.

[44] E. Öztürk, G. Hammouri, and B. Sunar, "Towards robust low cost authentication for pervasive devices", *6th Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2008*, pp. 170–178, 2008.

[45] R. Pappu, "Physical one-way functions", *Science*, vol. 297, no. 5589, pp. 2026–2030, 2001.

[46] G. Pólya, "Über den zentralen grenzwertsatz der wahrscheinlichkeitsrechnung und das momentenproblem", *Mathematische Zeitschrift*, vol. 8, no. 3, pp. 171–181, 1920.

[47] I. Rechenberg, "Evolutionstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution", *Step-Size Adaptation Based on Non-Local Use of Selection Information. In Parallel Problem Solving from Nature (PPSN3)*, 1973.

[48] IETF, "Internet security glossary", Internet Engineering Task Force, Glossary RFC 2828, 2000.

[49] M. Roel, "Physically unclonable functions: Constructions, properties and applications", PhD thesis, 2012.

[50] F. Rosenblatt, "The perceptron - a perceiving and recognizing automaton", Cornell Aeronautical Laboratory, Tech. Rep. 85-460-1, 1957.

[51] F. Rosenblatt, "Principles of neurodynamics: Perceptrons and the theory of brain mechanisms", in *Brain Theory: Proceedings of the First Trieste Meeting on Brain Theory, October 1–4, 1984*, G. Palm and A. Aertsen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 245–248.

[52] M. Rostami, M. Majzoobi, F. Koushanfar, D. S. Wallach, and S. Devadas, "Robust and reverse-engineering resilient PUF authentication and key exchange by substring matching", *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 37–49, 2014.

[53] U. Rührmair and J. Solter, "PUF modeling attacks: An introduction and overview", in *Design, Automation and Test in Europe Conference and Exhibition*, New Jersey: IEEE Conference Publications, 2014, pp. 1–6.

[54] U. Rührmair, H. Busch, and S. Katzenbeisser, "Strong PUFs: Models, constructions, and security proofs", in *Towards Hardware-Intrinsic Security: Foundations and Practice*, A.-R. Sadeghi and D. Naccache, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 79–96.

[55] U. Rührmair and D. E. Holcomb, "PUFs at a glance", *Design, Automation and Test in Europe Conference and Exhibition 2014*, pp. 1–6, 2014.

[56] U. Rührmair, C. Jaeger, and M. Algasinger, "An attack on PUF-based session key exchange and a hardware-based countermeasure: Erasable PUFs", in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7035 LNCS, Germany: Springer-Verlag, 2012, pp. 190–204.

[57] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions", *Proceedings of the 17th ACM conference on Computer and communications security - CCS '10*, p. 237, 2010.

[58] U. Rührmair, J. Solter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF modeling attacks on simulated and silicon data", *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1876–1891, 2013.

[59] U. Rührmair and M. Van Dijk, "PUFs in security protocols: Attack models and security evaluations", *Proceedings - IEEE Symposium on Security and Privacy*, pp. 286–300, 2013.

[60] S. Russell and P. Norvig, *Artificial intelligence: A modern approach*. Citeseer, 1995.

[61] T. Saha and V. Sehwag, "TV-PUF : A fast lightweight aging-resistant threshold voltage PUF", 2016.

[62] A. L. Samuel, "Some studies in machine learning using the game of checkers", *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[63] J. Seberry and X.-M. Zhang, "Highly nonlinear 0–1 balanced boolean functions satisfying strict avalanche criterion", in *Advances in Cryptology — AUSCRYPT '92: Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, December 13–16, 1992 Proceedings*, J. Seberry and Y. Zheng, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 143–155.

[64] F. Semiconductor. (2002). Basic analog and digital multiplexer design comparison, [Online]. Available: http://application-notes.digchip.com/009/9-12462.pdf.

[65] S. Skorobogatov, "Physical attacks and tamper resistance", in *Introduction to Hardware Security and Trust*, M. Tehranipoor and C. Wang, Eds., New York, NY: Springer New York, 2012, pp. 143–173.

[66] M. Soybali, B. Ors, and G. Saldamli, "Implementation of a PUF circuit on a FPGA", *2011 4th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2011 - Proceedings*, 2011.

[67] TCG, "TPM main part 1 design principles", Trusted Computing Group, Specification Version 1.2, 2011.

[68] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation", *Proceedings - Design Automation Conference*, pp. 9–14, 2007.

[69] S. Tajik, E. Dietz, S. Frohmann, H. Dittrich, D. Nedospasov, C. Helfmeier, J.-P. Seifert, C. Boit, and H.-W. Hübers, "A complete and linear physical characterization methodology for the arbiter PUF family", *IACR Cryptology ePrint Archive*, vol. 2015, p. 871, 2015.

[70] S. Tajik, E. Dietz, S. Frohmann, J.-P. Seifert, D. Nedospasov, C. Helfmeier, C. Boit, and H. Dittrich, "Physical characterization of arbiter PUFs", *Cryptographic Hardware and Embedded Systems*, pp. 493–509, 2014.

[71] S. Tajik, H. Lohrke, F. Ganji, J.-P. Seifert, and C. Boit, "Laser fault attack on physically unclonable functions", in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2015 Workshop on*, IEEE, 2015, pp. 85–96.

[72] M. Tehranipoor and C. Wang, *Introduction to hardware security and trust*. Springer Publishing Company, Incorporated, 2011.

[73] N. Wisiol, M. Margraf, T. Soroceanu, B. Zengin, C. Graebnitz, and M. Oswald, "Why attackers lose: Design and security analysis of arbitrary large XOR arbiter PUFs", unpublished.

[74] X. Xu and W. Burleson, "Hybrid side-channel / machine-learning attacks on PUFs : A new threat ?", *Design, Automation and Test in Europe Conference and Exhibition*, 2014.

[75] M. D. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, and I. Verbauwhede, "A lockdown technique to prevent machine learning on PUFs for lightweight authentication", *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 146–159, 2016.

Eidesstattliche Erklärung

Ich versichere hiermit, dass die Masterarbeit mit dem Titel *Machine Learning Attacks on Majority Based Arbiter Physical Unclonable Functions* von mir selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen genutzt wurden. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und ist unveröffentlicht.

Berlin, February 3, 2017

Manuel Oswald