A DESIGN STRATEGY TO IMPROVE MACHINE LEARNING RESILIENCY OF PHYSICALLY UNCLONABLE FUNCTIONS USING MODULUS PROCESS

by

Yuqiu Jiang

A Dissertation Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

in Engineering

at

The University of Wisconsin-Milwaukee

December 2023

ABSTRACT

A DESIGN STRATEGY TO IMPROVE MACHINE LEARNING RESILIENCY OF PHYSICALLY UNCLONABLE FUNCTIONS USING MODULUS PROCESS

by

Yuqiu Jiang

The University of Wisconsin-Milwaukee, 2023
Under the Supervision of Professor Weizhong Wang

Physically unclonable functions (PUFs) are hardware security primitives that utilize non-reproducible manufacturing variations to provide device-specific challenge-response pairs (CRPs). Such primitives are desirable for applications such as communication and intellectual property protection. PUFs have been gaining considerable interest from both the academic and industrial communities because of their simplicity and stability. However, many recent studies have exposed PUFs to machine-learning (ML) modeling attacks. To improve the resilience of a system to general ML attacks instead of a specific ML technique, a common solution is to improve the complexity of the system. Structures, such as XOR-PUFs, can significantly increase the nonlinearity of PUFs to provide resilience against ML attacks. However, an increase in complexity often results in an increase in area and/or a decrease in reliability. This study proposes a lightweight ring oscillator (RO)-based PUFs using an additional modulus process to improve ML resiliency. The idea was to increase the complex-

ity of the RO-PUF without significant hardware overhead by applying a modulus process to the outcomes from the RO frequency counter. We also present a thorough investigation of the design space to balance ML resiliency and other performance metrics such as reliability, uniqueness, and uniformity.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AMRO Alternating Modulus Ring Oscillator

APUF Arbiter Physically Unclonable Function

BRV Binary Response Value

CF Coin-Flipping

CMA-ES Covariance Matrix Adaptation Evolution Strategy

CRP Challenge Response Pair

FF Feed-Forward

FPGA Field Programmable Gate Array

GB Gradient Boosting

IPD Intertwined Programmable Delay Line

IRV Integer Response Value

LR Logistic Regression

LUT Look Up Table

ML Machine Learning

MRO Modulus Ring Oscillator

NMQ Non-monotonic Quantization

NN Neural Network

PDL Programmable Delay Line

PUF Physically Unclonable Function

RF Random Forest

RO Ring Oscillator

SRAM Static Random Access Memory

SVM Support Vector Machine

TERO Transient Element Ring Oscillator

VRO Virtual Ring Oscillator

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I am deeply grateful to my esteemed thesis advisor, Prof. Weizhong Wang. His exceptional guidance, unwavering support, and profound expertise have been instrumental in the successful completion of this research endeavor. Throughout this journey, Prof. Wang has provided invaluable mentorship, offering valuable insights, constructive feedback, and meticulous attention to detail. His patient and thoughtful supervision have played a pivotal role in shaping the direction, methodology, and overall quality of this thesis. I am truly fortunate to have had the privilege of working under his mentorship.

I would also like to extend my gratitude to my colleague Dr. Yangpingqing Hu. Working with Dr. Hu has been intellectually inspiring. His unique perspectives and contributions have played a significant role in shaping this work. I am grateful for his friendship and collaboration.

I am thankful to University of Wisconsin-Milwaukee for providing the necessary resources, and access to research materials enabling me to carry out this study effectively. The assistance provided by the staff and faculty of department of electrical engineering and computer science is gratefully acknowledged.

Lastly, I would like to express my heartfelt appreciation to my family and friends for their unwavering belief in me and their constant encouragement throughout this endeavor. Their understanding, patience, and unconditional support have been vital in keeping me motivated during the highs and lows of this research journey.

Chapter1

Introduction

1.1 Motivition

A key requirement for securing communication through open public networks is the ability to authenticate its counterpart at the other end of the communication channel. To block malicious network elements, a network node must validate the identities of the nodes with which it communicates. One such authentication method is to use unique hardware-dependent keys provided by.[1][2][3] The idea is to leverage non-reproducible variations during the manufacturing process to provide device-specific query-response pairs, which are also known as challenge-response pairs (CRPs). At the system level, the challenge is a bit string sent to the PUF embedded in the remote node as input. The PUF returns a response bit or a bit string as an answer. A remote node with an embedded PUF is authenticated if the response matches the expected result. Traditionally, it is impossible to predict or replicate subtle processing variations embedded in PUFs. There are several implementations of PUF. For example, Arbiter-PUF[4], ring oscillator PUF (RO-PUF)[5], transient element ring oscillator PUF (TERO-PUF)[6], SRAM-PUF[7], and butterfly-PUF[8]. Many PUF variants are vulnerable to machine-learning (ML)-based modeling attacks[9][10][11][12][13]. Researchers have proposed several implementations to improve ML resiliency. Examples include XOR-PUFs[14],

feed-forward PUF (FF-PUF), coin-flipping PUF (CF-PUF), and interpose PUFs[15]. However, many of the improved designs have been found to be vulnerable to more advanced ML attack techniques.[13][16][17]

1.2 Claims and Contribution

In this work, there are two contributions. First, a modulus-based RO-PUF (MRO-PUF) is proposed to improve the ML attack resiliency. MRO is a technique that increases the complexity within a PUF system by increasing the number of decision boundaries for RO's binary response. The idea is to generate a binary response by applying a modulus process to the integer response of the RO-PUF, which is typically the frequency-counter output. Thus, the complexity of the system can be increased and manipulated by varying the parameters of the modulus process, without significant hardware overhead. The proposed design is an alternative framework to NoPUF[18], which can be used to balance ML resiliency and reliability, while improving uniqueness. A recent study (NMQ-PUF[19]) has employed non-monotonic quantization (NMQ) to enhance the resilience of machine learning (ML) models. The NMQ method shares similarities with the modulus process and has demonstrated promising ML resilience outcomes with an application-specific integrated circuit (ASIC) implementation. However, the pro-posed NMQ-PUF fails to address one of the key reliability issues associated with the use of an aggressive quantization parameter, which is crucial when implementing the modulus parameters. To address this challenge, we propose an alternating modulus RO-PUF (AMRO-PUF) that balances the trade off between ML resilience and reliability. The AMRO-PUF can serve as an alternative paradigm for improving the controllability of the design process, thereby facilitating the achievement of desired performance outcomes. In this work, we focus on investigating the performance of

MRO based on configurable RO-PUF, such as the programmable delay line RO-PUF (PDL-RO-PUF) and intertwined programmable delay (IPD-RO-PUF). However, any strong PUF structure with numerical measurement, such as traditional RO-PUF and TERO-PUF, can be used as the base entropy source for the proposed modulus process.

Second, both the underlying RO-PUF structure and the proposed MRO structure were implemented on a Xilinx field programmable gate array (FPGA) to provide a comprehensive experimental demonstration of the potential improvement in machine learning resiliency.It is worth noting that FPGA implementations have inherent limitations in terms of place and routings when compared to other implementations such as ASIC. As a result, a thorough investigation of entropy sources implemented in FPGAs can provide valuable insights in terms of balancing the performance metrics.

1.3 Outline

The thesis is organized as follows: Chapter 2 provides background on the development on PUFs (section 2.1), definitions of performance metrics (section 2.2), and ML modeling attack methods against PUFs (section 2.3). Chapter 3 provides design strategies on MRO-PUF. In addition to the base MRO-PUF architecture (section 3.1), we also proposed a variant AMRO-PUF (section 3.2) and a uniformity optimization strategy (section 3.3). Implementation of the MRO-PUF is elaborated in chapter 4, where we also provided a hardware over head analysis to MRO-PUF and AMRO-PUF (section 4.3). The empirical result is demonstrated and discussed in chapter 5. Finally, chapter 6 concludes this thesis.

Chapter2

Preliminaries

2.1 Physically Unclonable Function (PUF)

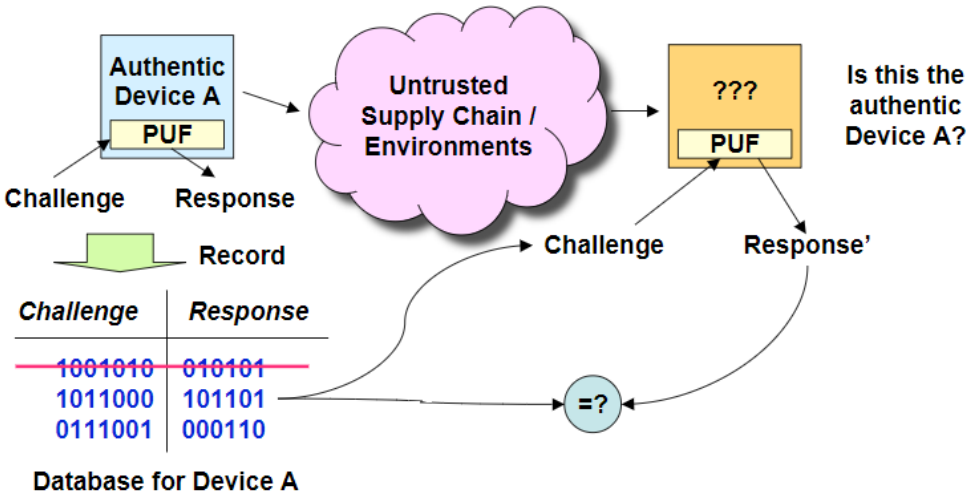


Figure 2.1: Illustration of authentication using PUF[14]

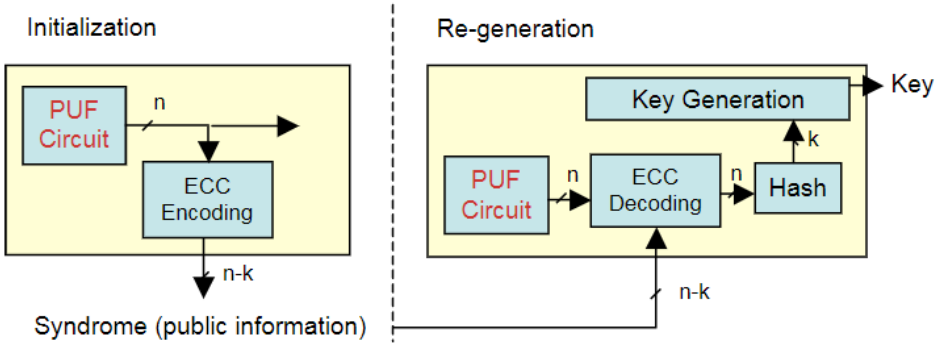


Figure 2.2: Illustration of secure key generation using PUF[14]

A Physically Unclonable Function (PUF) as first proposed by Pappu et al in 2001.[1], is a hardware security primitive that generates a unique and unpredictable response based on the physical characteristics of the underlying hardware. PUFs exploit the inherent variations in manufacturing processes to create unique identifiers or keys. PUFs have various applications in the field of security, authentication, and secure communication. Some common applications include: device authentication, secure key generation, anti-counterfeiting, secure bootstrapping. (see figure 2.1, figure 2.2)

One way to classify different types of PUFs is by distinguishing between weak PUFs and strong PUFs. Weak PUFs are characterized by their CRP (Challenge-Response Pairs) space, which grows linearly with incresing available resources. An example of a weak PUF is the traditional RO-PUF, where each CRP requires a unique pair of RO components. Despite resource limitations, weak PUFs often provide better randomness and are more resilient against ML attacks.

On the other hand, strong PUFs have a CRP space that typically grows exponentially with area, offering practically unlimited CRPs. However, when resources are shared between CRPs, strong PUFs may exhibit inter CRP dependencies, making them vulnerable to ML

modeling attacks. The arbiter PUF and configurable RO-PUF are examples of strong PUFs [20].

Another method of classifying different types of PUFs is based on the entropy extraction method. One approach is to compare delays between different propagation lines. The method base on the manufacturing variations in delays in gates and routes. When a racing condition is intentionally created, the outcome is although unpredictable with only knowledge on design but reproducible. Both APUF and RO-PUF are good examples for such delay-based PUFs.

Another method involves reading the state of memory after initialization. The variations of memory cells provide desirable reproducible response. A example for memory-based PUF is SRAM-PUF.[7]

The following sections discuss the foundational concepts and principles of popular PUF base and state-of-the-art variants, with a specific focus on the delay-based method. Intermediate delay measurement is a crucial requirement for MRO-PUF.(see section 3.1) However, it is worth noting that all variants can offer comparable performance metrics. There is a set of performance metrics are common evaluated across all PUFs design. (see section 2.2)

2.1.1 Arbiter-PUF (APUF)

The APUF is an early incarnation of PUFs, first proposed by Gassend et al. in 2002, shortly after the introduction of PUFs [5]. The APUF circuit, depicted in Figurefigure 2.3, is also known as a non-monotonic delay circuit. It consists of multiple delay stages, where each stage is controlled by a challenge bit. Each challenge bit selects one of the two delay paths within each stage. The output is generated based on the relative delay difference between the two configured paths at the end of the delay line.

With each additional delay stage, the Challenge-Response Pair (CRP) space of the APUF doubles, making it a strong PUF. However, the APUF faces challenges due to the common delay element between CRPs and the additive nature of the total delay in a configuration. It has been reported to be vulnerable to Machine Learning (ML) attacks [10][13].

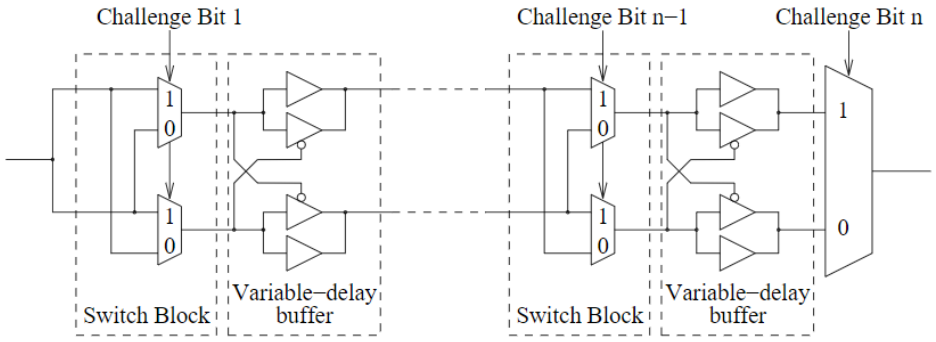


Figure 2.3: Structure of APUF[5]

2.1.2 Ring-Oscillator PUF (RO-PUF)

RO-PUF is first proposed by Suh and Devadas in 2007[14] which is referred as the traditional RO-PUF though out this paper. Figure 2.4 illustrate the structure of traditional RO-PUF. A RO-PUF circuit consists of multiple identically laid-out delay loops, also known as ring oscillators. Each ring oscillator is a simple circuit that oscillates at a specific frequency. Due to manufacturing variations, each ring oscillator oscillates at a slightly different frequency. To generate a fixed number of bits, a predetermined sequence of oscillator pairs is selected, and their frequencies are compared to produce an output bit. The output bits obtained from the same sequence of oscillator pair comparisons will differ from chip to chip. Since the oscillators are identically laid out, the frequency differences are determined by manufacturing variations, and an output bit is equally likely to be a one or a zero if random variations dominate.

The traditional RO-PUF demonstrates a innovative way to reliably harness the entropy using ROs. At the time of its publication, RO-PUF offered an low-cost implementation on FPGAs.

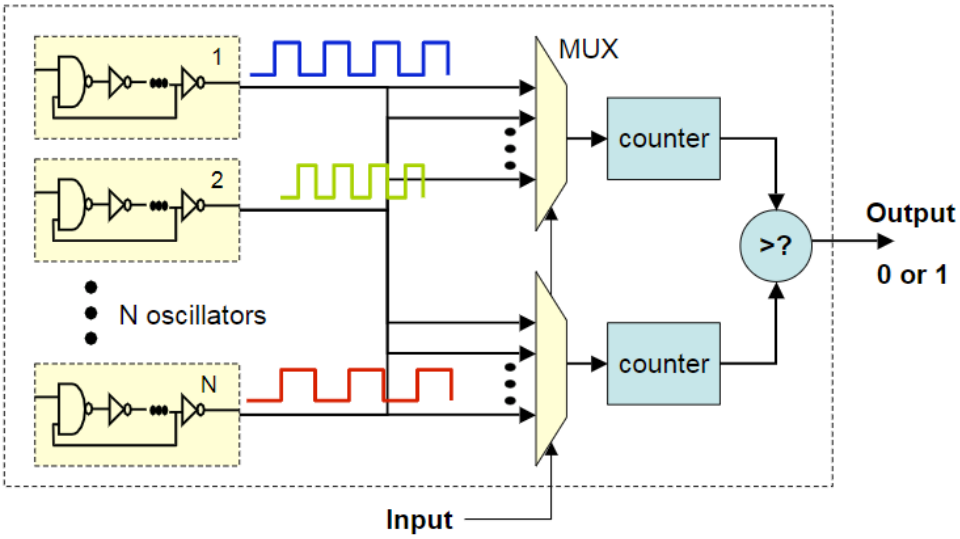


Figure 2.4: Structure of RO-PUF[14]

Programmable Delay Line RO-PUF (PDL-RO-PUF)

In contrast to transitional RO-PUF, there is a branch of RO-PUF focuses on utilizing a configurable RO instead of RO bank. Such RO-PUF variant is also known as PDL-RO-PUF. The concept of PDL was first proposed by Majzoobi et al.[21] The PDL was first implemented with APUF to better its adpattation to FPGA. Where APUF was first focussing on ASIC[14]. Later Habib et al adopted the idea of PDL to RO-PUF in 2013,[22] which later Feiten et al built up on[23].

Habib's PDL-RO-PUF introduced a structure of configurable RO which consists of an even number of inverter stages and a NAND or NOR gate forming the loop, as shown in figure 2.5. Each inverter stage is configured by a number of challenge bits. An enable signal

is connected to the NAND or NOR gate to enable or disable oscillation. The response bit is generated by comparing the oscillation counter between two configurable PDL-RO with the same challenge. Habib's PDL-RO-PUF demonstrate the RO-PUF can be modified to strong PUF with the configurable RO structure enables a much larger CRP space comparing to Suh's RO bank. Habib also discussed the advantage of using FPGA LUTs to accommodates multiple challenges bits to configure a single inverter stage.

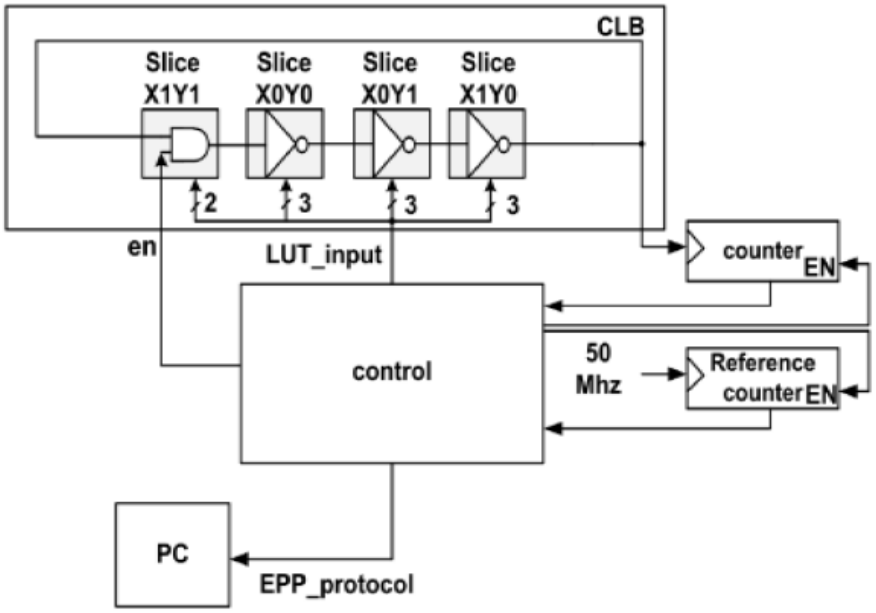


Figure 2.5: Architecutre of Habib's RO-PUF with PDL [22]

However, Feiten et al discovered that the PDL structure implemented in FPGA has systematic bias in the delay path, due to the unconfigurable routing paths. Feiten's contribution to the PDL-RO-PUF is to introduced a sequntial sampling strategy to PDL-RO-PUF. The idea is to sequentially sampling the same configurable RO with different configuration instead of having two RO instances with the same configuration. (see figure 2.6) The sequntial sampling minimizes the routing biases between different RO instances. Feiten also investigated the internal structure of FPGA's LUT primitive and discussed possible strategies

to two keep routing symmetrical between measurement phases(see figure 2.7) The challenge with such 2-pass scheme is that the temporal variation between the two measurement is introduced. With the symmetrical measurement constraint the selection of configuration of PDL become laborious and reduces the CRP space.

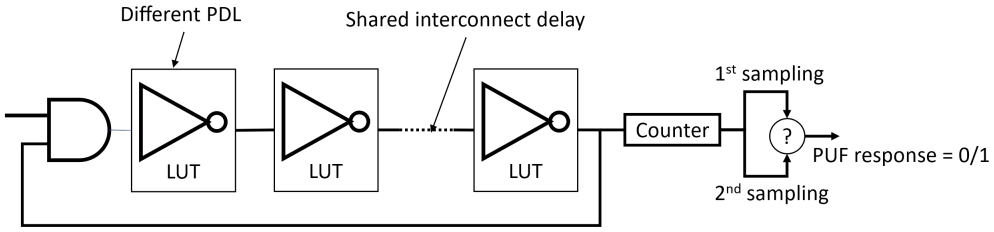


Figure 2.6: Illustration of Feiten's sequntial sampling with PDL-RO-PUF

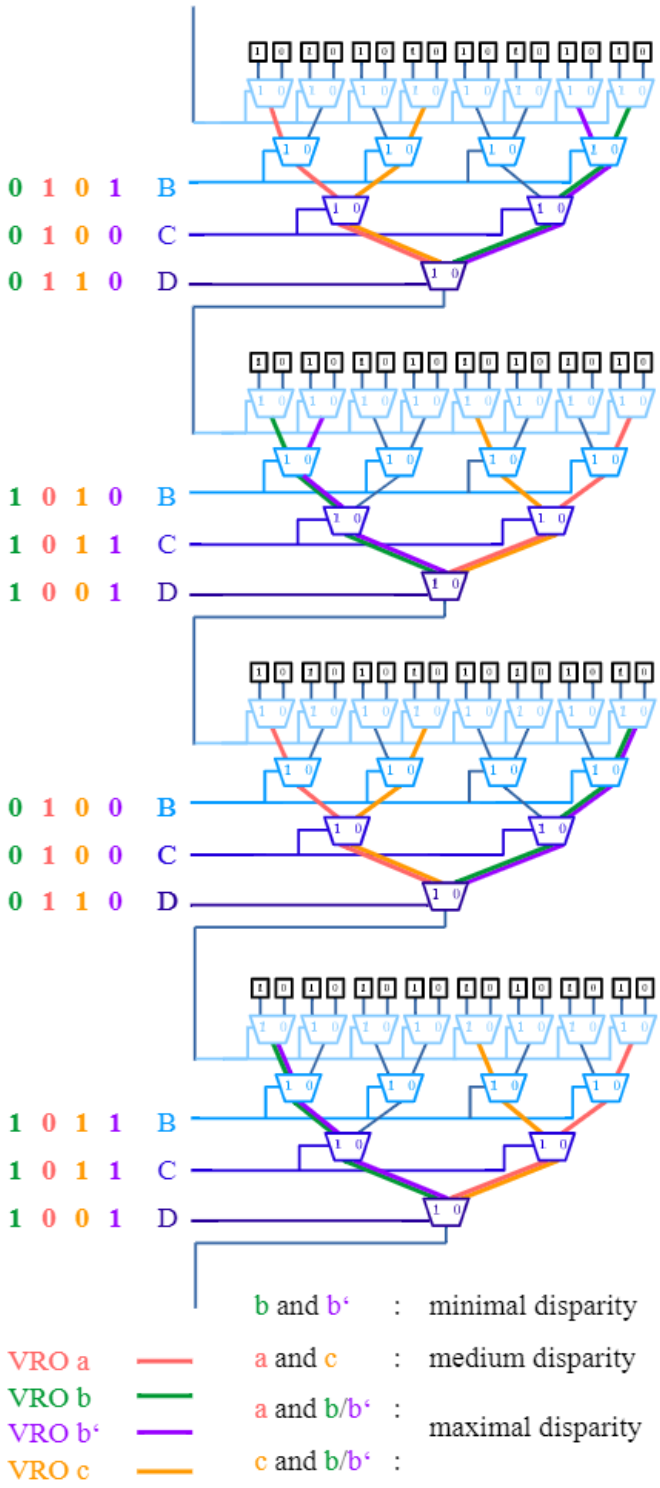


Figure 2.7: Feiten's path disparity analysis of four configurable LUTs of a PDL-RO.[23]

Intertwined Programmable Delay RO-PUF (IPD-RO-PUF)

In our early work let by Hu,[24] we proposed a novel IPD-RO-PUF structure to further minimizes the systematic biases challenges with PDL-RO-PUF. IPD-RO-PUF uses a IPD structure to replace teh LUT stage in PDL-RO-PUF to minimize systematic bias. The IPD-RO and IPD inverter stages are shown in figure 2.8 and figure 2.9, respectively. The IPD inverter has two LUTs, implemented in a single LUT stage. The idea is to have a two-pass process in addition to sequential sampling (two runs) inherited from PDL-RO-PUF[23] to cancel the systematic bias in the LUTs. The two-phase process toggles the switch bit of the IPD stage such that two nominally symmetrical delay lines are measured in each pass. The assumption is that a systematic bias is common in both phases. Therefore, by comparing these two measurements for each run. The majority of systematic bias was canceled. Systematic bias leads to a correlation between delays and the corresponding challenges. Furthermore, a reference RO was implemented in the IPD-RO-PUF to reduce the impact of disturbances from voltage and thermal noise.

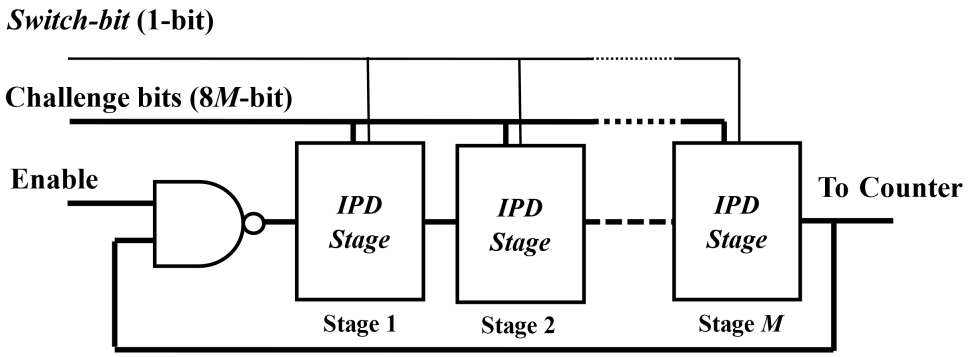


Figure 2.8: Structure of IPD-RO-PUF[24]

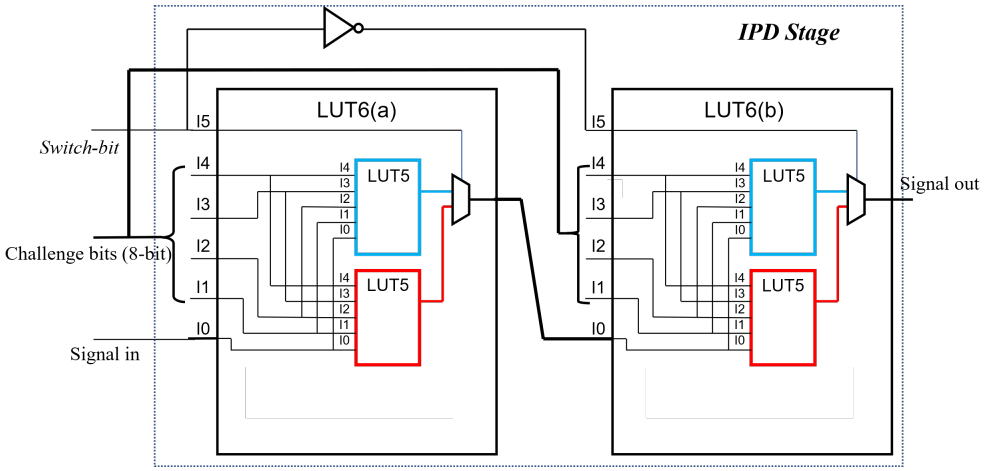


Figure 2.9: The intertwined programmable delay (IPD) stage.[24]

The IPD-RO-PUF demonstrate a promising performance such as uniformity, uniqueness, and reliability. Similar to Habib's and Feiten's PDl-RO-PUF, it leverages the FPGA's LUT to increase the complexity of each inverter cell. However the underlying additive delay line structure still show disadvantages against ML attacks.

2.2 Standard Performance Index

2.2.1 Uniformity

Uniformity is a measure of the binary outcome distribution of PUFs. For an ideal PUF, the binary response should have the same probability of generating 1s and 0s. Uniformity is defined by equation (2.1)

$$H = \frac{\sum_i^M R_i}{M} \tag{2.1}$$

where, R_i denotes the response bit. M is the number of samples. The uniformity ranged from 0 to 1. The ideal value is 50%, indicating that there are same number of samples for the response bit being 1 and 0. For a measured uniformity either smaller or larger than 50%, this suggests that the probability of 0 or 1 being generated is larger than the other. Uniformity dictates the minimum ML prediction accuracy of the PUF.

2.2.2 Bit-aliasing

Bit aliasing is a metric that demonstrates the biases among different PUF instances. The definition of bit aliasing is introduced in [6]. Bit-aliasing measures the likelihood of a group of PUFs generating the same response to the same challenge, as shown in equation (2.2).

$$A_l = \frac{1}{k} \sum_{i=1}^{k} r_{i,l} \times 100\% \tag{2.2}$$

where $r_{i,l}$ is the lth binary bit of the response from PUF instance i.

2.2.3 Uniqueness

Uniqueness metrics are defined as the variations in the responses of different instances derived from the same PUF design. Ideally, different instances of the same PUF design would produce uncorrelated responses.

The traditional definition of uniqueness is shown in the following formula. equation (2.3)

$$U = \frac{2}{N(N-1)} \sum_{j=1}^{N} \sum_{i=j+1}^{N} \frac{HD(\mathbf{R}_i, \mathbf{R}_j)}{K} \times 100\% \tag{2.3}$$

where N denotes the number of PUF instances. $\mathbf{R}_i$ is the response vector from the ith instance. $HD(\cdot)$denotes the Hamming distance. The ideal value is 50%. K denotes the number of responses in each response vector.

2.2.4 Correlation

In addition to traditional uniqueness, the correlation in CRPs must also be investigated to address correlated signature bits [25][26]. Feiten's Correlation evaluation is defined as follows:

For each response bit, a correlation matrix is computed by comparing the outcomes of the n signature bits of each device i with one another, resulting in $\frac{(n\cdot(n-1))}{2}$ pairings:

$$\begin{pmatrix} cor^i_{1,2} & cor^i_{1,3} & ... & cor^i_{1,n} \\ & cor^i_{2,3} & ... & cor^i_{2,n} \\ & & ... & ... \\ & & & cor^i_{n-1,n} \end{pmatrix} \tag{2.4}$$

with:

$$cor^i_{j,k} = \begin{cases} 1, & \text{if } sig_{i,j} = sig_{i,k} \\ -1, & \text{otherwise} \end{cases} \tag{2.5}$$

The $cor^i_{j,k}$ values for each pariing are then summed up over m devices:

$$cor_{j,k} = \frac{1}{m}\sum_{i=1}^{m} cor^i_{j,k} \tag{2.6}$$

The ideal distribution of the $cor_{j,k}$ is a normal distribution with a mean of 0. The standard deviation of the distribution is related to the number of samples tested. Theoretically, if an infinite number of devices are tested, all calculated correlations should be 0.

2.2.5 Reliability

The reliability of PUFs is the probability of obtaining the same responses when the same challenge bit patterns are given to the PUF. This is one of the key performance metrics of PUFs. Reliability is commonly defined by (equation (2.7)) based on the measured data[27].

$$R = \frac{|\frac{M}{2} - \sum_i^M \mathbf{R}_i|}{\frac{M}{2}} \tag{2.7}$$

where M denotes the number of samples collected for the same challenge. Ri is the binary response value of each sample. The reliability ranges from 0 to 1, where 0 is the least reliable and 1 is the most reliable.

2.3 Machine Learning Modeling Attack

Parallel to the advancement of PUFs, extensive research has been conducted on modeling attack strategies. Currently, there is no universally accepted framework for evaluating the

resilience of PUFs against machine learning (ML) attacks. To address this, we aim to incorporate various ML modeling techniques, including both traditional and cutting-edge approaches, which have demonstrated varying levels of success in modeling different PUF variants. The ML modeling methods considered in this study are as follows:

- Logistic Regressionsection 2.3.1
- Support Vector Machinesection 2.3.2
- Gradient Boostingsection 2.3.3
- Random Forestsection 2.3.4
- Neural Networksection 2.3.5
- Reliability-based Modelingsection 2.3.6

2.3.1 Logistic Regression

LR is a well investigated ML frame work which was originally developed ad popularized by Berkson[28]. LR has been reported successful at modeling PUF such as APUF and XOR-APUF.[10]

For PUFs with single-bit outputs, each challenge $C = b_1 \dots b_k$ is assigned a probability $p(C, t|\vec{w})$ of generating an output $t \in \{-1, 1\}$. The vector $\vec{w}$ represents the relevant internal parameters of the PUF, such as runtime delays. The probability is calculated using the logistic sigmoid function $\sigma(tf)$, where $f(\vec{w})$ is a function parameterized by $\vec{w}$. The sigmoid function is defined as $p(C, t|\vec{w}) = \sigma(tf) = (1 + e^{-tf})^{-1}$. The decision boundary, determined by $f = 0$, separates regions with equal output probabilities. To position the boundary for a

given training set $\mathbb{M}$ of Challenge-Response Pairs (CRPs), the parameter vector $\vec{w}$ is chosen to maximize the likelihood of observing the set or minimize the negative log-likelihood. (see equation (2.8))

$$\vec{m} = argmin_{\vec{w}} l(\mathbb{M}, \vec{w}) = argmin_{\vec{w}} \sum_{(C,t)\in\mathbb{M}} -ln(\sigma(tf(\vec{w}, C))) \tag{2.8}$$

As there is no analytical solution for equation (2.8). The solution is typically obtained through it gradient equation (2.9).

$$\nabla l(\mathbb{M}, \vec{w}) = \sigma_{(C,t)\in\mathbb{M}} t(\sigma(tf(\vec{w}, C)) - 1)\nabla f(\vec{w}, C) \tag{2.9}$$

2.3.2 Support Vector Machine

SVM similar to LR performs as a binary classification method with a decision boundary. However, SVM differs from LR in that it can sovlve for systems that is not linearly separable. Research has shown that SVM has been successful in modeling APUFs.[29].

During the learning phase, known training examples are mapped into a higher-dimensional space to facilitate the classification task. This technique is used to handle classification problems that are not linearly separable in the original input space. By mapping the data to a higher-dimensional space, the learning algorithm aims to find a good separating hyperplane that can accurately classify the data.

The goal is to find a hyperplane that maximizes the margin, which is the largest possible distance between input vectors belonging to different classes. The inputs that lie closest to

the separating hyperplane are known as support vectors. The construction of the separating hyperplane involves two parallel supporting hyperplanes that pass through the support vectors. The distance between these supporting hyperplanes is referred to as the margin.

To build an SVM, the objective is to maximize the margin while minimizing the classification error. These objectives are balanced by a regularization parameter, which determines the trade-off between the margin and the error.

Trained SVMs heavily depend on the self-inner product of the mapping function, also known as the kernel. The kernel is evaluated on both the support vectors and the challenge or test inputs that need to be classified.

2.3.3 Gradient Boosting

Gradient Boosting is an ensemble learning technique that combines multiple weak models to create a stronger predictive model. It is based on the concept of boosting, where weak models are sequentially trained to correct the mistakes made by the previous models.[30] GB is considered as an effective ML modeling against PUF.[12]

In GB, the weak models are typically decision trees. The algorithm works by iteratively adding new decision trees to the ensemble, with each subsequent tree aiming to correct the errors made by the previous trees. This is achieved by fitting each new tree to the residual errors of the ensemble's predictions.

The concept of gradients plays a key role in GB. After each iteration, the algorithm calculates the gradients of a loss function with respect to the ensemble's predictions. These gradients provide information about the direction and magnitude of the errors. The next

weak model is then trained to minimize the loss function by following the negative gradient direction.

To prevent the model from overfitting, regularization techniques such as tree depth limits and learning rate adjustments are often applied. Additionally, subsampling techniques such as stochastic gradient boosting can be used to further improve generalization.

The final predictions of the ensemble are obtained by summing the predictions of each weak model, with each model assigned a weight based on its performance during the training process.

2.3.4 Random Forest

Random Forest is a powerful and widely used ensemble-based machine learning algorithm that combines the predictions of multiple decision trees.[31] It can be used for both classification and regression tasks. The algorithm is also been considered as one of the effective ML modeling method against PUF.[32]

At its core, Random Forest creates a collection of decision trees, each trained on a random subset of the training data. During training, each tree is built by repeatedly splitting the data based on different features, aiming to create decision boundaries that effectively separate different classes or predict continuous values. The randomness introduced in the training process helps to create diverse and independent decision trees.

During prediction, each tree in the Random Forest independently generates a prediction based on the input data. For classification tasks, the final prediction is determined by majority voting, where the class with the most votes from the trees becomes the predicted

class. For regression tasks, the final prediction is typically the average of the predictions from the individual trees.

One of the key advantages of Random Forest is its ability to handle high-dimensional data set with a large number of features. The algorithm automatically selects a subset of features at each node of the decision trees, reducing the impact of irrelevant or noisy features and improving the overall performance. This feature selection process also helps to mitigate the risk of overfitting.

2.3.5 Neural Network

Neural networks, also known as artificial neural networks, are a class of machine learning models inspired by the structure and functionality of the human brain. They have been successfully applied in various domains such as image classification, natural language processing, and time series prediction.

At a high level, a neural network is composed of interconnected nodes called artificial neurons or "units." These units are organized into layers, typically consisting of an input layer, one or more hidden layers, and an output layer. Each unit receives input signals, performs a computation, and produces an output signal. The hidden layers help the network learn complex patterns and representations from the input data.

The connections between units are represented by weights, which determine the strength of the relationship between inputs and outputs. During the training process, the network adjusts these weights using algorithms like backpropagation, which calculates the gradient of the loss function with respect to the weights. This allows the network to minimize the difference between its predicted outputs and the true values in the training data.

Neural networks can have different architectures, such as feedforward networks, recurrent networks, and convolutional networks. Feedforward networks are the most common type, where information flows from the input layer to the output layer without cycles. Recurrent networks, on the other hand, have connections that form loops, allowing them to model sequential data. Convolutional networks are designed for processing grid-like data such as images, by using convolutional layers for feature extraction.

In recent years, deep learning has become synonymous with neural networks, thanks to the development of deep neural networks with many hidden layers. Deep learning has achieved remarkable success in various tasks, often surpassing human performance in areas like image recognition and natural language understanding.

Neural networks have certain advantages, such as their ability to learn complex representations, handle large datasets, and generalize well to unseen data. However, they also have limitations, including the need for substantial computational resources, the potential for overfitting, and the difficulty of interpreting their internal workings.

2.3.6 Reliability-based Modeling

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is an evolutionary optimization algorithm specifically designed for solving continuous optimization problems. It is particularly effective in cases where the objective function is expensive to evaluate or exhibits non-linear, non-convex, or multimodal characteristics.

CMA-ES belongs to the class of Evolution Strategies (ES), which are stochastic, population-based optimization algorithms inspired by biological evolution. The main idea behind CMA-ES is to adaptively estimate and update the covariance matrix of the multivariate distribution

that represents the population of candidate solutions.

Instead of explicitly maintaining a fixed mutation operator like in traditional Evolution Strategies, CMA-ES dynamically adjusts the shape and orientation of the mutation distribution based on the past success or failure of the candidate solutions. This adaptation is done by updating the covariance matrix during the evolutionary process.

The CMA-ES algorithm works as follows:

1. Initialization: Start by randomly generating an initial population of candidate solutions. Choose an initial covariance matrix and a step size.
2. Sample Solutions: Generate new candidate solutions by sampling from a multivariate Gaussian distribution with mean vector and covariance matrix.
3. Evaluating Solutions: Evaluate the fitness or objective function value for each candidate solution.
4. Selection: Select the best-performing solutions based on their fitness values.
5. Covariance Matrix Update: Update the covariance matrix based on the selected solutions. This step includes computing the covariance matrix adaptation and adjusting the step size.
6. Termination Criterion: Repeat steps 2-5 until a termination criterion is met, such as a maximum number of iterations or a satisfactory fitness value.

CMA-ES is known for its strong robustness and efficiency in handling complex optimization problems with large-scale variables. It can adapt effectively to the problem landscape, automatically adjusting the search distribution to explore promising regions. Moreover, it

does not require any gradient information, making it suitable for black-box optimization scenarios.

Chapter3

MRO-PUF

3.1 MRO Architecture

As shown in figure 3.1, the proposed MRO structure consists of two main components: a entropy source and a modulus process module. The entropy source takes in the challenge information and generate a device-specific signature. To compliant with the modulus process, the entropy source is requires to be capable of generating a integer response value (IPV). One common form of IPV is the frequency measurement of a RO. In this work, the IPD-RO-PUF is considered as a promissing entropy source candidate. Therefore most of the discussion in this work references the IPD-RO-PUF (see implementation in section 4.1). However, the proposed architecture is compatible with any PUF variant that is capable generate IPV as an intermediate response.

The modulus process module calculate the modulus of the measurement from the entropy source and generate a binary output. The modulus process provides additional complexity which consequently enhancing the ML resiliency.

At the beginning of the MRO-PUF operation, the configurable RO takes in the challenge bit pattern and starts to oscillate. Then, the counter measures the oscillation cycles

for both phase 1 and 2 of the RO. The RO and counter, shown as two parallel processes in figure 3.1, were implemented on the same hardware with two sequential measurement processes (see section 2.1.2 and section 2.1.2). Subtracting the counter readings from the two phases yields an integer value. The subtraction between the two phases minimizes the common delay bias from different routings in each LUT, which also provides a numerical response value. Therefore, a sequential sampling technique [23] is feasible. By randomly selecting two different challenge bit patterns in the challenge space, an integer response can be generated by comparing the responses from the two different challenges. In contrast to the conventional binary response, the generated integer response, called the integer response value (IRV), is used in this study. IRV reflects the quantified delay variation caused during manufacturing. If the process is run multiple times with the same pair of challenge bit patterns, the IRV would vary slightly because of measurement noise, such as thermal and voltage fluctuations, which generally fit into a normal distribution. This contributes to a degradation in reliability, which will be examined in Section IV. The oscillation frequency variances among different RO pairs, chosen by different challenge bit patterns, also generally fit into a normal distribution. This was because the randomness of the manufacturing process followed a normal distribution. Therefore, the IRV variation can be described at two different levels. In this study, the standard deviation of the IRV for a specific challenge pattern is called the specific sigma. The standard deviation of the IRV from all the challenges is called the general sigma. The measurement noise is correlated with a specific sigma. Manufacturing variation correlates with general sigma. The sigma ratio of a general sigma to a specific sigma reflects the reliability of the RO-PUF. From the experimental data collected from the Zynq 7000 FPGA, we found that IPD-RO-PUF had a sigma ratio of approximately 20, which resulted in a reliability performance of approximately 98%. PDL-RO-PUF has a sigma ratio of approximately 40, which results in a reliability of approximately 99%.

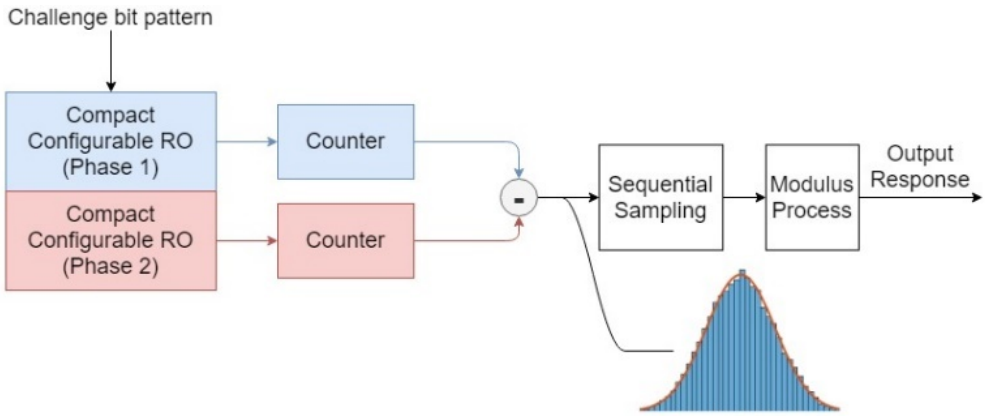


Figure 3.1: Architecture of MRO-PUF with IPD-RO-PUF as entropy source

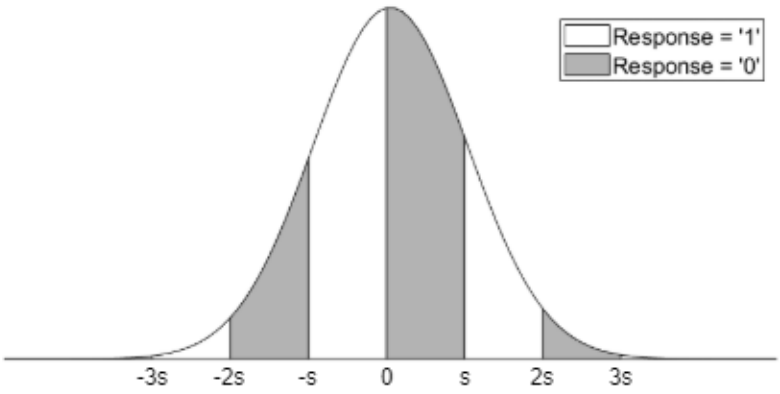


Figure 3.2: Modulus Decision Boundary on Integer Response Value Distribution (modulus factor = s)

Traditionally, the binary response is obtained by comparing the oscillation speed or frequency between the two ROs. If the binary response output of the PUF has two classes: '0' and '1', then the theoretical decision boundary of the two binary classes is when the two compared RO have the same propagation delay. In other words, the IRV of PUF was 0. Considering, the total propagation delay of a configurable RO is the sum of several inverter delays: Each inverter delay is controlled using a challenging bit. Therefore, such traditional methods are vulnerable to machine-learning attacks, such as logistic regression (LR) or NN attacks[33]. This claim is supported by the test results presented in . A modulus response

boundary system is proposed to improve the performance against machine learning attacks. In contrast to the binary response, the idea here is to split the IRV space into several regions, with a binary response value assigned to each region in an alternating manner (see figure 3.2). The conversion from IRV to a BRV is shown in equation (3.1). The goal is to achieve better machine learning attack resilience by increasing the complexity of the system.

$$B = \lfloor \frac{r}{m} \rfloor \mod 2 \tag{3.1}$$

where, B is the binary response value. r is the IRV from response counter. m is the modulus factor that determines the width of each response region. The smaller the response region, the more difficult it is to be modeled. Subsequently, the modulus decision boundary (see figure 3.2) can be defined as follows:

$$s = km, k \in \mathbb{Z} \tag{3.2}$$

3.2 Alternating MRO (AMRO)

With an increase in the number of modulus decision boundaries, the MRO is at a risk of reduced reliability. The smaller the modulus factor, the greater the number of decision boundaries introduced. In this study, an additional structure is proposed to balance the reliability and resiliency against ML. The idea is to use multiple modulus factors, such that each zone has a different width. As the simplest case, the two modulus factors can be selected based on their ML resiliency and reliability in their single MRO performance. Considering each modulus factor as a controllable factor, it can be leveraged to achieve an optimal trade

off between the machine learning resiliency and reliability.

In terms of defining the alternating policy, we propose using the parity bit generated by the input challenge pattern in a 2 modulus factors design. This policy ensures that the likelihood of using each modulus factor is approximately 50%. The structure of the AMRO is illustrated in figure 3.3. The RO and sampling processes were identical to those used for the MRO structure. An alternating modulus structure requires two modulus processes based on different modulus factors. The binary outcome of both modulus processes is selected by the parity bit generated from the challenge-bit pattern. This process does not require a significant amount of resource overhead, while introducing an additional control- lable modulus factor. With the two-modulus process utilizing two different modulus factors, the reliability and ML accuracy can be balanced by fine-tuning the modulus factors.

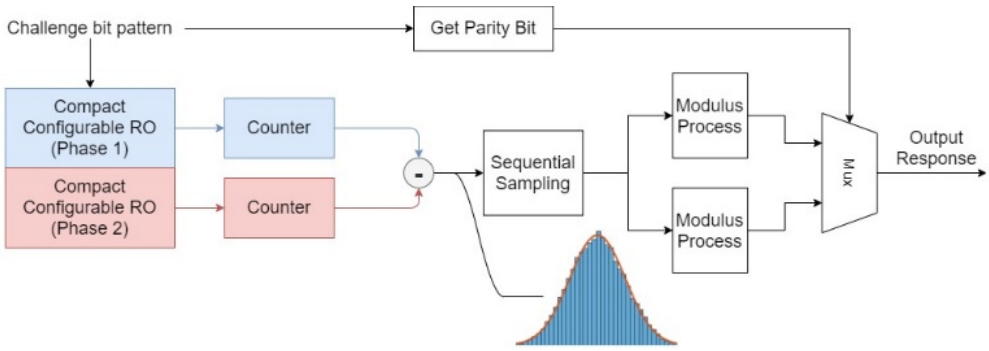


Figure 3.3: Structure of AMRO

3.3 Uniformity Optimization

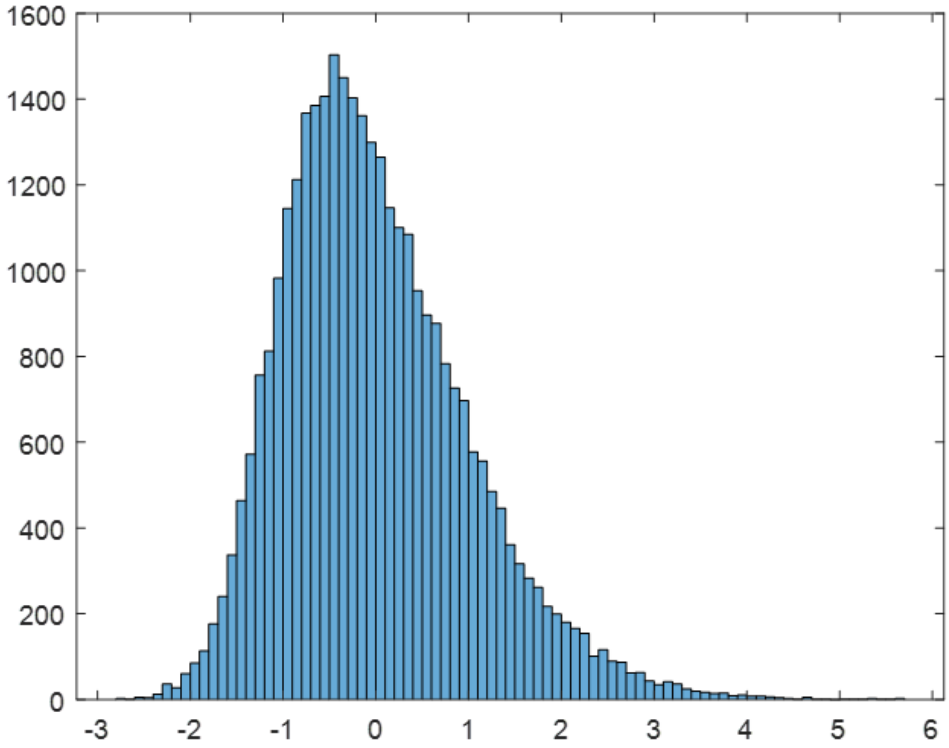


Figure 3.4: Simulated asymmetrical integer response population

When investigating the uniformity of the MRO, the difference in uniformity with varying modulus factors was negligible. The assumption is that an RO model with a symmetrical integer response distribution. However, considering a PUF with an asymmetrical distribution, applying the modulus process could have either a positive or negative impact on the uniformity, depending on the modulus factor and the shape of the distribution. To demonstrate the impact of MRO on a PUF with an asymmetrically distributed integer response, a simulated integer response population (see figure 3.4) with a Poisson distribution was created and tested using the modulus process. Note that the simulated distribution is intentionally

different from the symmetric experimental data. Therefore, the modulus factor in the simulation scenario does not match with the scaling with the experimental data. The modulus pro- cess is identical to other discussions in this work. As shown in figure 3.5, the uniformity with a large modulus factor is 56%. When the modulus factor decreases to 32, the uniformity worsens to approximately 60%. Furthermore, the uniformity reaches an almost ideal uniformity value when the modulus factor decreases to 10. We found that when the modulus factor was sufficiently small, the difference in the number of samples between the two adjacent bins significantly decreased. Therefore, it is expected that by reducing the modulus factor, ideal uniformity can be achieved. However, reducing the modulus factor always poses a risk of rendering PUF unreliable.

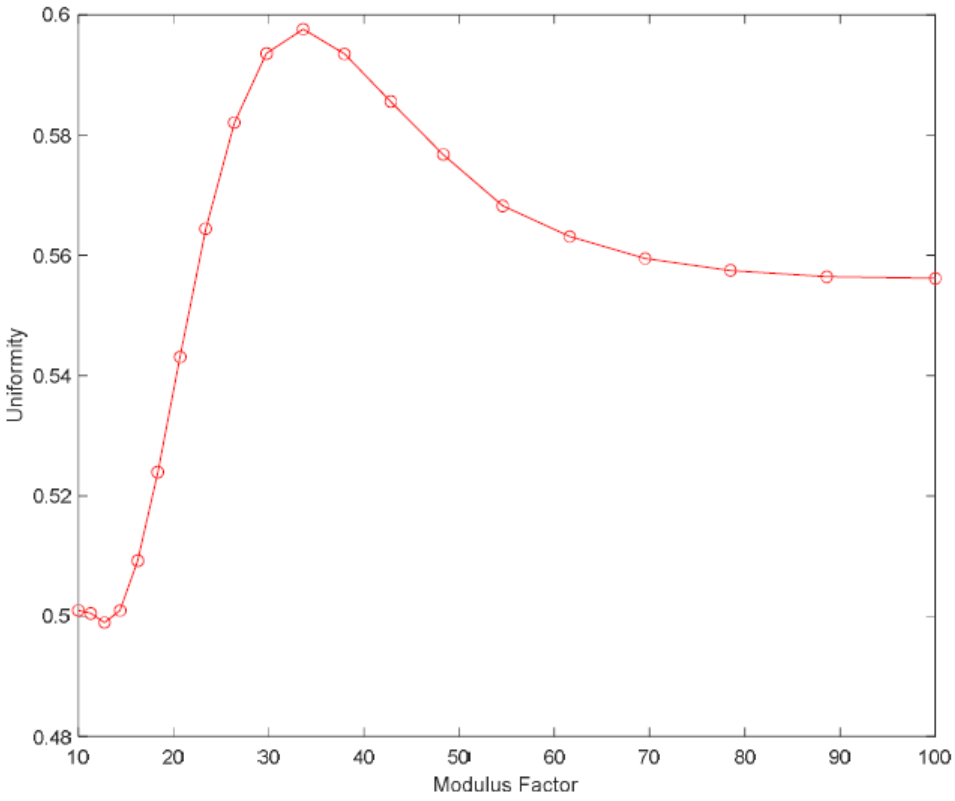


Figure 3.5: Uniformity of asymmetrical integer response with varying modulus factor

To mitigate the reliability-uniformity trade off concern, a policy derived from the MRO design is proposed to ensure optimal uniformity while increasing the number of decision boundaries. Considering adding a decision boundary to be an iterative process, the initially added decision boundary would split the whole integer response population into two decision regions with '0' and '1' assigned to each region. In terms of uniformity, the optimal decision boundary should split the region into two regions with equal populations. Such decision boundaries can be determined using the median of the population. (see equation (3.3)).

$$B_0 = median(\mathbb{S}) \tag{3.3}$$

where B_0 denotes the optimal decision boundary for the initial decision boudnary. $\mathbb{S}$ is the set of all sampled integer response.

Adding the decision boundary to each region can be treated as a process that is similar to adding the initial decision boundary. (see equation (3.4).

$$B_n = median(\mathbb{S}_n + \epsilon) \tag{3.4}$$

where B_n denotes the nth decision boundary introduced. $\mathbb{S}_n$ is a subset of $\mathbb{S}$. $\mathbb{S}_n$ contains all the samples in the region where the decision boundary B_n is added. ϵ is a random value that is either a positive or negative small value with equal possibility. The purpose of adding ϵ is to balance the numbers of 1s and 0s. The proposed decision boundary ensures near-50% uniformity in every decision region in which the boundary is added. In addition, introducing each optimal uniformity boundary is a standalone process that has a minimal impact on uniformity, which can be leveraged in terms of considering the number of decision

boundaries and selecting the decision region to add the boundary. As shown in figure 3.6, an example using the proposed decision boundary is demonstrated. The uniformity remained at approximately 50% with an increased number of decision boundaries.

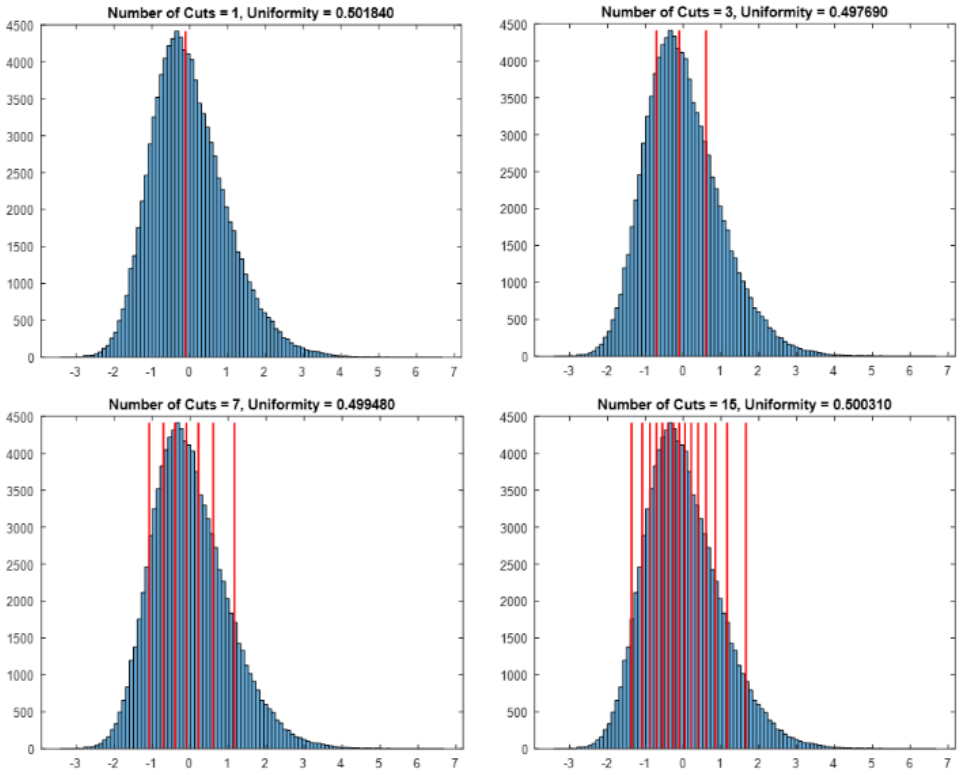


Figure 3.6: Uniformity performance with increasing number of decision boundaries

Chapter4

Implementaiton

To demonstrate the machine learning resilience performance of the proposed MRO PUF design, both PDL-RO-PUF and IPD-RO were implemented as the underlying hardware core. The experimental setup was implemented using a Xilinx Artix-7 FPGA. The evaluated performance metrics included uniformity, uniqueness, reliability, and machine learning attack resilience under different modeling attack techniques.

4.1 Underlying Entropy Source

The IPD-RO design (see figure 4.1) uses three input signals: challenge-bit patterns, enable, and switch control. The output was an oscillating signal. Challenge bit patterns configure each cascading inverter stage in RO. Each inverter stage contributed to the oscillation frequency. The enable signal is a binary control signal that enables and disables oscillations through a NAND gate. The switch control signal is also a binary control signal that selects between the phase 1 and phase 2 operations of the PUF. The IPD-RO structure utilizes primitive LUT elements for the inverter stages in RO. (see Figure 9) Each LUT is programmed to function as a single configurable inverter. Owing to the multiplexing nature of LUT cells, the LUT input challenge-bit pattern controls the propagation delay of the inverter, which is

selected from several independent candidate inverters. For example, a 3-bit input challenge pattern for an LUT cell has eight possible combinations. Each combination corresponds to a delay parameter independent of any of the other seven possibilities. If we consider a traditional design, as shown in Figure 2, each challenge bit controls the selection of one inverter among the two. In this case, the total propagation delay of a 3-bit challenge input setting is the summation of the three selected inverters. The eight possible delays are not independent but linearly dependent on the three delay stages.

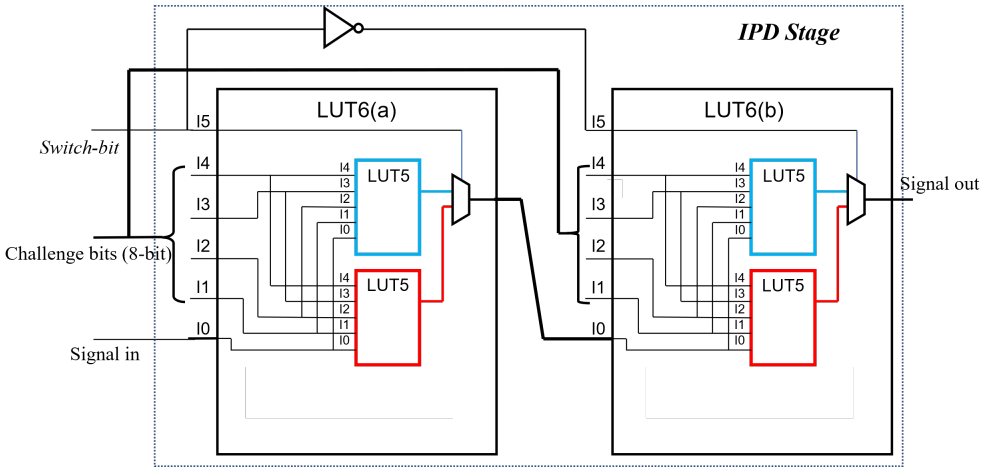


Figure 4.1: The stage structure of IPD-RO-PUF

As shown in figure 4.2, each stage within the IPL-RO-PUF is formed by two cascaded LUT6s (six-bit input LUT module from Xilinx FPGA library). Among the six inputs of each LUT, I1 acts as the input of the inverter and is connected to the adjacent LUT output, I2–I5 takes the 4-bit challenge bit pattern, and I6 takes the switch bit (which sets phase 1 or phase 2 of the operation). At any given time, one LUT takes the switch bit and the other LUT takes the inversion of the switch bit. This arrangement always makes the two LUTs in one stage to be in opposite modes. ROs with 1, 2, 4, and 8 of these stages were implemented. These correspond to PUFs of 4, 8, 16, and 32 bits of the challenge bit, respectively. Because an odd number of inverters are required in RO to enable oscillation, a 2-input NAND gate

based on LUT5 was added to the RO. One of its inputs is connected as part of the ring, whereas the other input is used as a logic-high-enabling pin for control purposes.

Voltage and temperature are the two main factors affecting RO frequency. To decrease the variation caused by these two factors, a reference RO was implemented to calibrate the output from PUF RO. The reference ring is always parallel to the PUF ring. The readings from the PUF rings were calibrated using the readings from the reference ring. Because the reference ring and PUF ring both experience the same voltage, temperature, and clock variations, the common variation is cancelled. Reference RO and PUF RO were instantiated using the same customized design IP. To better cancel the measurement-related noises, we constrained the reference RO and PUF RO to be in the same clock and power regions of the FPGA.

PDL-RO-PUF was implemented using the same RO structure, as shown in figure 4.2. By setting the switch control bit to a constant value, each intertwined stage behaves the same as the inverter stage in PDL-RO-PUF. Therefore, the hardware was configured as a PDL-RO-PUF.

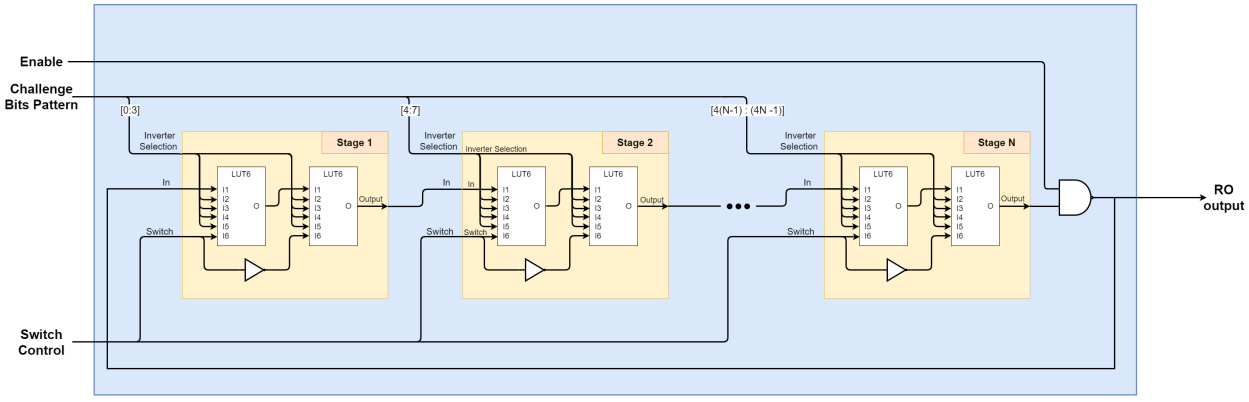


Figure 4.2: Implementation of RO as entropy source

4.2 Implementaiton of MRO-PUF

The MRO is implemented as a post-process after the RO integer response values are extracted. In this study, a single MRO structure was demonstrated with 40-bit IPD-RO and 40-bit PDL-RO configurations. Both were implemented in a Xilinx FPGA Zynq 7000 SoC.

The response data collected from the hardware were paired with the corresponding challenge bit strings to form CRPs. The challenge patterns are in the form of binary vectors with lengths that match the RO size. The integer response values were recorded and used in the modulus and screening process. Note that when the PUF is in service, the modulus process is performed completely within the FPGA fabric, with no integer value leaving the FPGA. Leaking the integer response value to a potential attacker could make the system vulnerable to machine learning modeling attacks.

The response from the same challenge bit string may vary owing to the measurement noise, even with noise-cancelling reference ROs. (See chapter 5) To that end, each challenge is tested in multiple trials to evaluate the reliability. The estimated reliability is described in detail in chapter 5.

4.3 Hardware Overhead Analysis

MRO is enabled by utilizing the counter in the underlying RO-PUF. The hardware complexity depends on the modulus factor chosen. In table 5.5, a hardware analysis on both MRO and AMRO is provided. The data shown in table 5.5, is based on experimentation on Xilinx Zynq 7000 FPGA with 10-bit modulus factor. The unit of the data is the number of LUTs

used in the Xilinx FPGA. No block memory units or DSP slices are used in this experiment. Therefore, they are omitted from the table.

Table 4.1: Hardware overhead of MRO and AMRO (measured in number of LUTs)

Modulus factor implementation	MRO with modulus factor = X	AMRO with modulus factor = [X,Y]
Single bit wire	0	9
Constant	147	283
Variable signal	350	710

For both MRO and AMRO, three different modulus processes are implemented to demonstrate the relation between the level of modulus factor control and hardware overhead. The first MF implementation aims to minimize the hardware footprint. In the case of MRO, a single wire connected to the one bit of the frequency counter can provide the binary response with modulus factor of the order of 2. In terms of AMRO, some logic to generate the parity bit of the challenge pattern is required. Therefore, there is a small amount of hardware overhead. The second MF implementation provides full modulus factor space. The modulus factor is set as a constant before deployment. Such implementation is suitable for applications which do not need to modify the properties of the PUF in real-time. The constant modulus factor used in this experiment is the largest prime number in the MF space which is 1021. The third MF implementation provides the maximum control over MF. The full modulus factor space is available. In addition, the modulus factor can be changed in real time. With maximizing the controllability, it requires the most hardware overhead amongst the three implementations. Such configuration is ideal for applications that have different usage for the PUF. For example, there may be scenarios that prioritize reliability while others prioritize ML resiliency. For the second and third implementations, the AMRO

requires roughly double the hardware resources than MRO.

Chapter5

Investigation on MRO Performance

The proposed MRO was a variant of the RO-PUF family. Therefore, the standard performance metrics for PUFs are applicable. The standard performance metrics considered in this study include uniformity, bit aliasing, uniqueness, correlation, reliability, and machine learning resilience.

The MRO test bench was implemented on a Xilinx Artix-7 FPGA (see figure 5.1). The operation was managed using Zynq PS (processing system). For a single-frequency reading, Zynq first sends the challenge-bit pattern to the BRAM input. BRAM then passes this challenge to the RO. After the counter was cleared, RO was enabled for a predetermined period of time. The counter counts the number of oscillation cycles. The counter value is then read by the BRAM and passed on to the PS.

Both PDL-RO-PUF and IPD-RO-PUF require aggregating multiple frequency readings to the final IRV. In this test bench, the calculation of the IRV and modulus process of the MRO were postprocessed off the FPGA chips.

In this work, we implemented both PDL-RO-PUF and IPD-RO-PUF on five Xilinx Zynq FPGAs. On each FPGA, we implemented 32 instantiations of each type of RO-PUF as the underlying entropy source. We collected 65,536 unique CRPs from each instantiation.

The collected experimental data were used for performance metrics analysis.

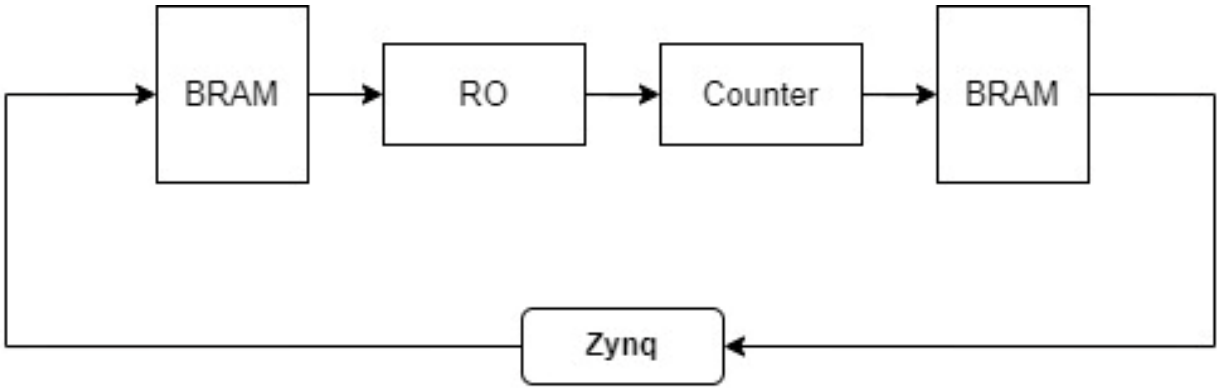


Figure 5.1: Block diagram of Xilinx Zynq testbench

5.1 Standard Performance Metrics Analysis

5.1.1 Uniformity

As defined in equation (2.1), the uniformity was computed with varying modulus factors. As shown in figure 5.2, the uniformity of the experimental data was close to the ideal scenario of 50%. However, we do find that the uniformity either slightly improves or deteriorates depending on the underlying entropy source. (Where PDL has slight improvement with more aggressive modulus factor and IPD has slight decrease)

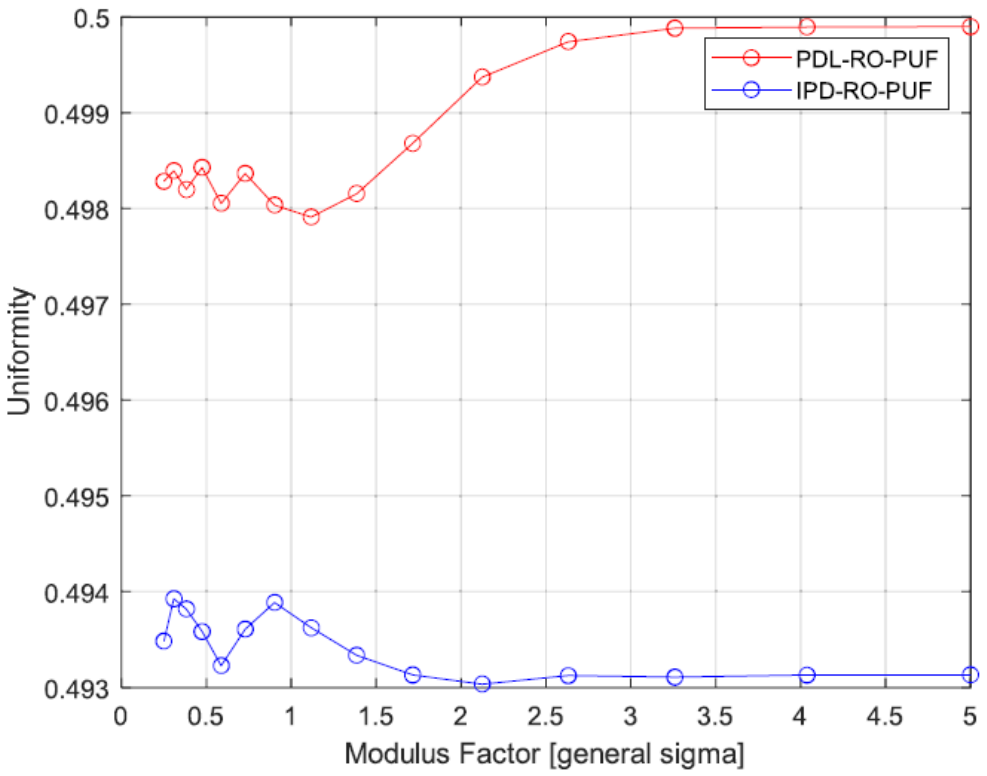


Figure 5.2: Uniformity with PDL-RO-PUF and IPD-RO-PUF, modulus factors up to 5 times the general sigma

5.1.2 Bit-aliasing

To investigate the impact of MRO on bit aliasing, bit-aliasing is computed (see equation (2.2) for 160 instances of both PDL-RO-PUF and IPD-RO-PUF. Each instance generates 65,536 CRPs, and the distribution of bit aliasing is observed for each RO-PUF configuration. Specifically, the analysis includes the following configurations: ideal, traditional (without modulus process), and MRO with the modulation factor (MF) varying from 0.2 to 2. The ideal distribution is computed using a random number generator that generates the same number of CRPs as the other configurations, and its mean is close to 50%, with a standard deviation

related to the sample size. In this case, the standard deviation of the ideal distribution is 3.95. Table 5.4 shows that without the modulus process, the mean is close to the ideal, but both PDL and IPD have worse standard deviation. Notably, PDL-RO based implementation has over 7 times the standard deviation compared to the ideal configuration. However, with the modulus process, the standard deviation of the bit aliasing distribution improves. Both PDL-RO and IPD-RO based implementations achieve near-ideal bit aliasing performance when the MF is lower than 0.5.

Table 5.1: Distribution comparison of bit aliasing analysis

	PDL-RO-PUF		**IPD-RO-PUF**	
configuration	**mean**	**std**	**mean**	**std**
Ideal	50.000	3.950	50.000	3.950
MF=0.2	50.051	3.939	50.012	3.948
MF=0.425	50.015	3.964	50.079	3.955
MF=0.65	50.032	4.177	50.114	3.955
MF=0.875	50.047	7.168	50.163	3.947
MF=1.1	50.039	12.722	50.146	3.975
MF=1.325	50.017	18.392	50.122	4.064
MF=1.55	50.007	23.226	50.120	4.186
MF=1.775	50.005	27.045	50.144	4.357
MF=2	50.008	29.868	50.131	4.510
w/o MF	50.020	34.495	50.134	4.832

5.1.3 Uniqueness

Uniqueness is defined as equation (2.3). As shown in figure 5.3, the conventional uniqueness of PDL-RO-PUF was significantly improved by the addition of modulus process. As the uniqueness reduced from over 70% to approximately 50% at 0.6 modulus factor. In contrast, IPD- RO-PUF, which has good uniqueness performance natively, benefits very little from the MRO.

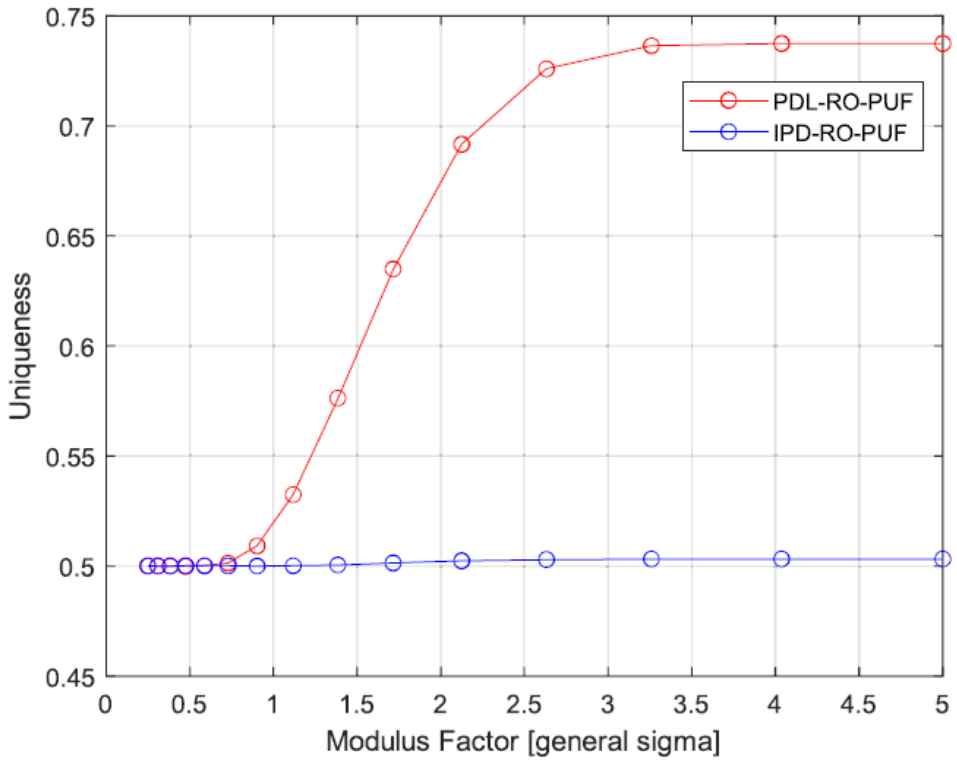


Figure 5.3: Conventional uniqueness with PDL-RO-PUF and IPD-RO-PUF, modulus factors up to 5 time sthe general sigma

To further investigate the performance of uniqueness, the distribution of the Hamming distance between each PUF instance was also investigated. (see table 5.2) Similar to the testing setup of bit aliasing, three configurations are included: ideal, traditional (without

modulus process), and MRO. The IPD and PDL based implementation have significantly larger standard deviation without modulus process. The PDL-RO based implementation also shows a shift of mean, which is aligned with the observation in figure 5.3. Both IPD-RO and PDL benefit with the addition of modulus process. With MF at approximately 0.5, both PDL-RO and IPD-RO archived close to ideal distribution, where the mean is approximately 50% and the standard deviation is approximately 0.196.

Table 5.2: Distribution comparison of uniqueness analysis

	PDL-RO-PUF		**IPD-RO-PUF**	
configuration	**mean**	**std**	**mean**	**std**
Ideal	50.000	0.196	50.000	0.196
MF=0.2	49.998	0.196	49.999	0.198
MF=0.425	50.002	0.196	50.000	0.195
MF=0.65	50.037	0.209	50.001	0.197
MF=0.875	50.720	0.654	50.000	0.203
MF=1.1	52.943	1.591	50.004	0.279
MF=1.325	56.493	2.393	50.018	0.627
MF=1.55	60.542	2.833	50.038	1.353
MF=1.775	64.406	2.991	50.068	2.299
MF=2	67.640	3.000	50.095	3.242
Traditional	73.634	2.848	50.156	5.344

5.1.4 Correlation

The Feiten's correlation analysis is defined as equation (2.4) equation (2.5) equation (2.6).

The addition of the modulus process is expected to improve the correlation metric of a PUF. Similar to the distribution analysis of bit-aliasing and uniqueness. The mean and standard deviation for Feiten's correlation are shown in table 5.3. The ideal mean is approximately 0 and the standard deviation is 0.0791 considering the sample size. Without the modulus process, both PDL-RO and IPD-RO based implementations exhibit larger standard deviation which are 0.4785 and 0.1336 respectively. However, with an MF below 0.875, the correlation distributions are close to the ideal. This indicates that both IPD-RO and PDL-RO based implementations can benefit from the addition of MRO.

Table 5.3: Distribution comparison of Feiten's correlation analysis

	PDL-RO-PUF		**IPD-RO-PUF**	
configuration	**mean**	**std**	**mean**	**std**
Ideal	0.0002	0.0791	0.0002	0.0791
MF=0.2	0.0000	0.0790	0.0002	0.0791
MF=0.425	0.0001	0.0792	0.0001	0.0790
MF=0.65	-0.0001	0.0792	0.0003	0.0791
MF=0.875	0.0001	0.0810	0.0002	0.0792
MF=1.1	0.0002	0.1039	0.0003	0.0793
MF=1.325	-0.0002	0.1589	0.0003	0.0798
MF=1.55	-0.0001	0.2278	0.0003	0.0833
MF=1.775	0.0006	0.2964	0.0000	0.0912
MF=2	-0.0003	0.3729	0.0003	0.1022
Traditional	-0.0004	0.4785	0.0000	0.1336

5.1.5 Reliability

For each CRP, the reliability was calculated using equation (2.7). The overall reliability for a PUF design across all challenges was calculated by taking the mean of all measured challenge-specific reliability.

Because the cause of unreliability is usually measurement noise crossing the decision boundary, an increased number of decision boundaries from the modulus process would have a negative impact on reliability.

Considering the impact of MRO, the reliability degrades with a decreasing modulus factor, as expected. However, as shown in the following figure (see figure 5.4), the reliability was still over 95% when the modulus factor was above 2.

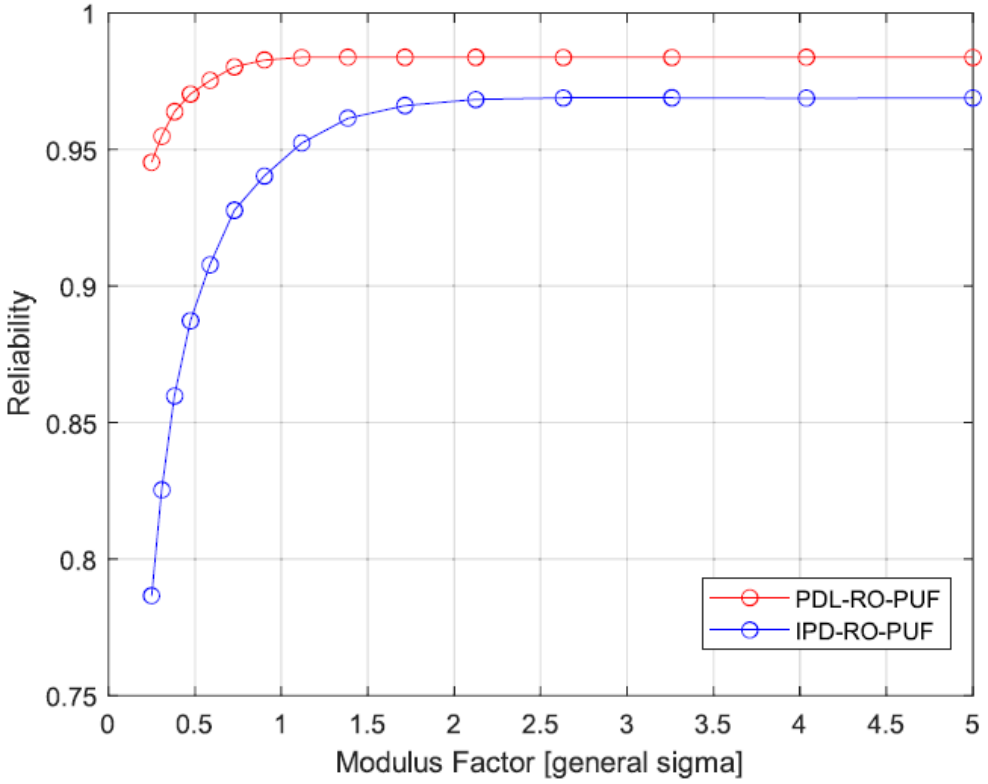


Figure 5.4: Reliability without screening, modulus factors up to 5 time sthe general sigma

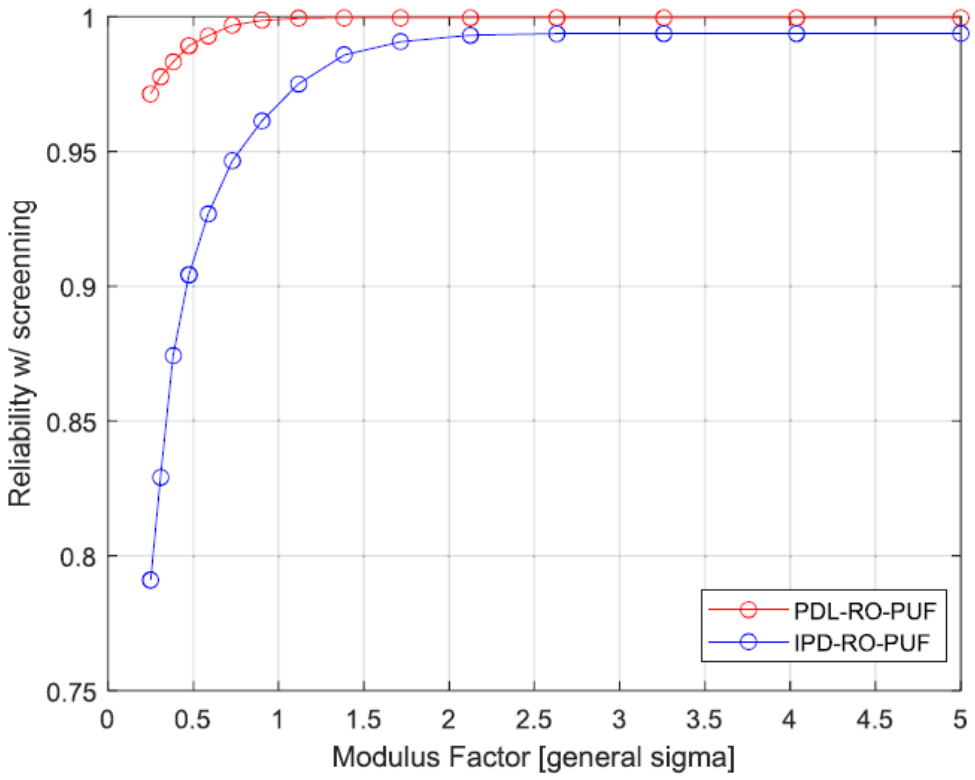


Figure 5.5: Reliability without screening, modulus factors up to 5 time sthe general sigma

Furthermore, several methods can be implemented to improve reliability, such as error-correction codes and CRP screening protocol. In this study, we experimented with a simple screening policy to demonstrate the potential improvement in reliability while sacrificing the challenge space. The screening policy is defined as follows. When a response appears to have a distance smaller than 1.5 specific sigma values to any decision boundary, it is considered unreliable. Hence, the corresponding challenge-response pairs were discarded. The reliability with and without the screening policy is shown in figure 5.4 and figure 5.5, respectively. The data reduction rates are shown in figure 5.6.

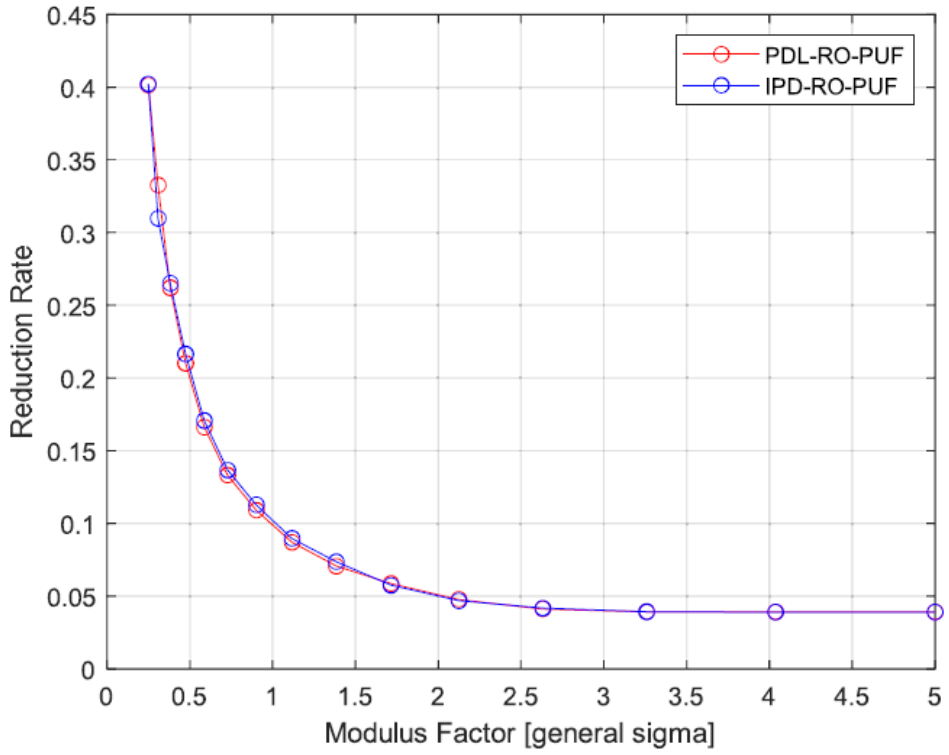


Figure 5.6: Data reduction rate from the screening, modulus factors up to 5 time sthe general sigma

5.2 Machine Learning Analysis

5.2.1 CRP-based Modelling

To demonstrate resilience against machine-learning attacks, both PDL-RO-PUF and IPD-RO-PUF were used as entropy sources. We used 8 RO instances from 2 Xilinx FPGAs for the ML analysis to provide generality across different FPGAs and RO instances. (2 RO instances by 2 FPGAs for PDL and 2 by 2 for IPD) We collected 65,536 CRPs from each RO instances with same set of challenge patterns. 70% of the data is used for training. The

rest is used for testing and cross-validation. The data volume is sufficiently large compared to other recent work on ML resiliency analysis on PUF. (see Table 3) The hardware used for the ML testbench is the following.

- CPU: Intel Core i7-6700K
- RAM: 32GB
- GPU: Nvidia GeForce RTX 2080

The software testbench is implemented with python 3.8.10. The detailed ML training configuration is shown in Table 2. Note that we used parameter tuner for both SVM and NN training to increase the confidence in the testing result which resulted in increment in training time. The other ML methods performed reasonably well with the default parameters. With the provided configuration, the average training time for a single set of CRPs considering all ML method is approximately 40 minutes. During each training session, the SVM, NN and GB take up most of the training time.

Table 5.4: Distribution comparison of bit aliasing analysis

ML method	**Average training time [sec]**	**Configuration**
LR	2.68	sklearn 0.24.0[34]
		default parameters
SVM	1146.33	sklearn 0.24.0 w/ grid search
		C=1
		gamma = [scale, 0.1, 0.01]
		kernel = rbf
GB	384.48	sklearn 0.24.0
		n_estimators=300
		learning_rate=0.7
		max_depth=5
RB	58.84	sklearn 0.24.0
		default parameters
NN	666.93	keras 2.4.3 [35] with
		kerastunner 1.0.2 RandomSearch
		num_layers = 4
		weights_per_layer = [min=32, max=256]
		activation =[relu, tanh, sigmoid]
		learning_rate = [min=1e-5, max=1e-2]
		loss_function = binary_crossentropy

Each modeling method was tested against three different MRO configurations. The first configuration is the single-modulus-factor MRO (figure 5.7 and figure 5.10). It is obvious

that MRO has the potential to reduce the ML attack accuracy when the modulus factor is below 2. Furthermore, the ML attack accuracy can be reduced to approximately 50%, which indicates the design under test is completely unpredictable when the modulus factor is below 1. The second and third configurations are AMRO with different modulus-factor policies. The second configuration follows a modulus policy, which has two alternating modulus factors that differ by a factor of two. (figure 5.8 and figure 5.11) The third configuration follows a policy that has one alternating modulus factor fixtured on 1 and sweeping the other. (figure 5.9 and figure 5.12) The fixtured modulus factor is selected based on the result from the first configuration (figure 5.7 and figure 5.10), where 1 general sigma is the maximum modulus factor achieving 50% prediction accuracy. AMRO policies provide options for designers to fine-tune the balance of the aforementioned performance metrics.

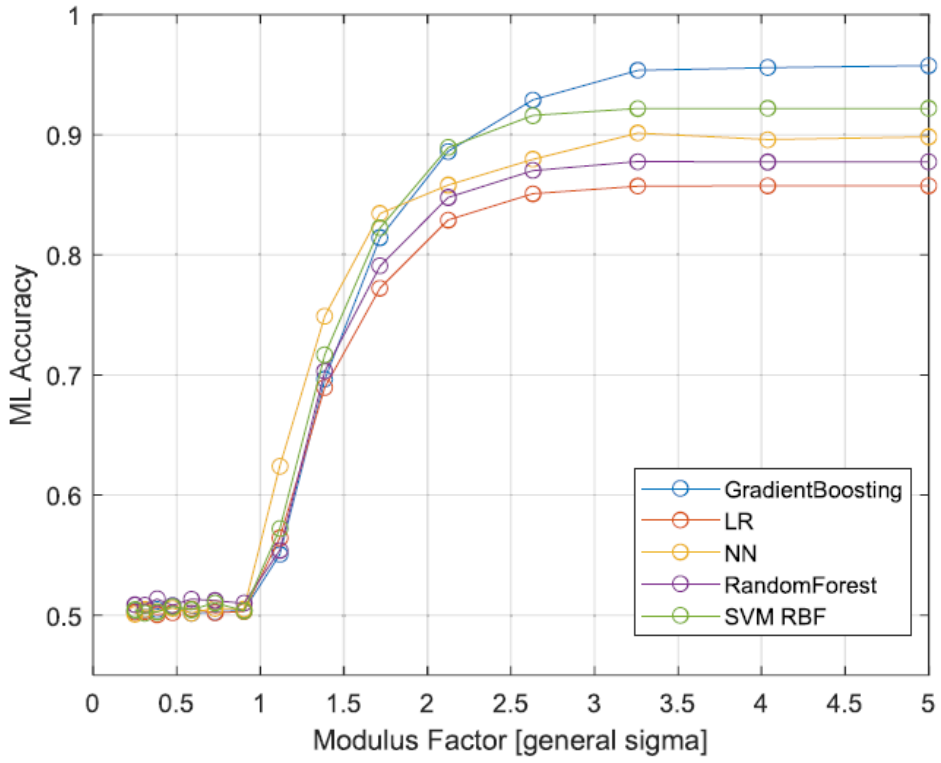


Figure 5.7: ML attack accuracy analysis on PDL-PO-PUF using single modulus factor MRO

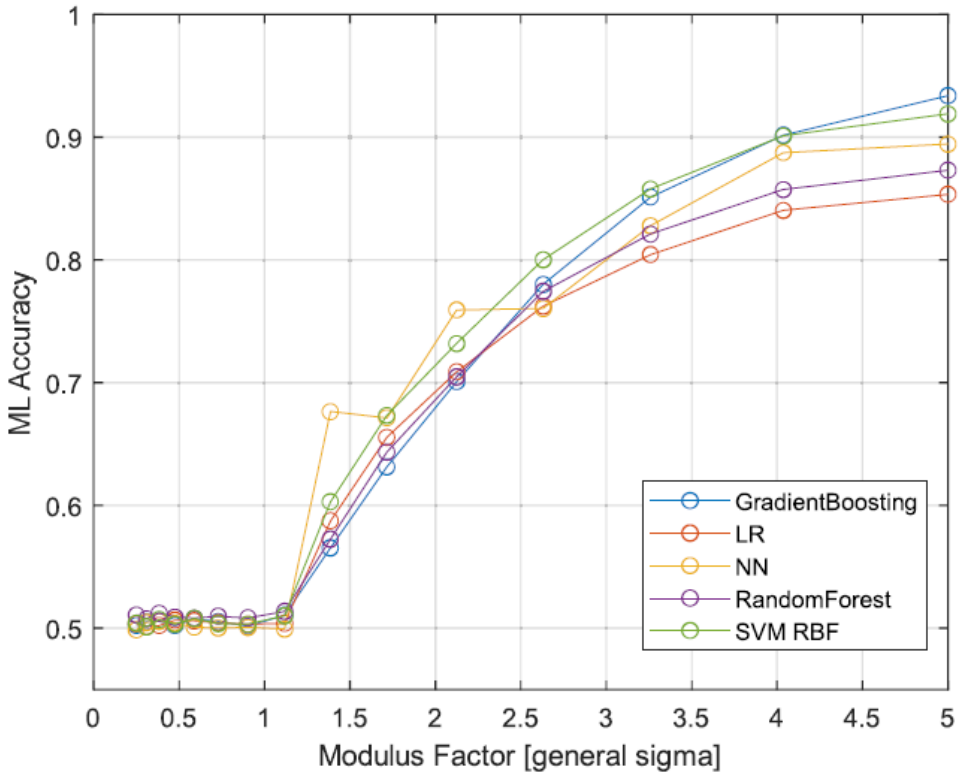


Figure 5.8: ML attack accuracy analysis on PDL-RO-PUF using AMRO with a policy of two modulus factor different by a factor of 2

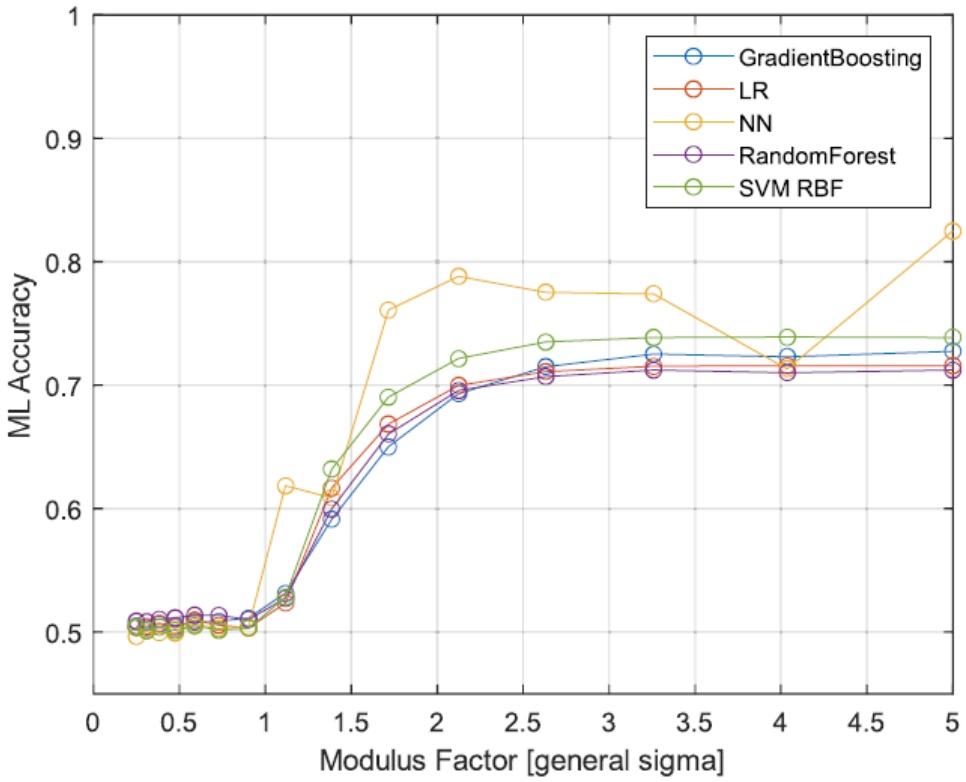


Figure 5.9: ML attack accuracy analysis on PDL-RO-PUF using AMRO with a policy of one modulus factor fixture on 1 while varying the other

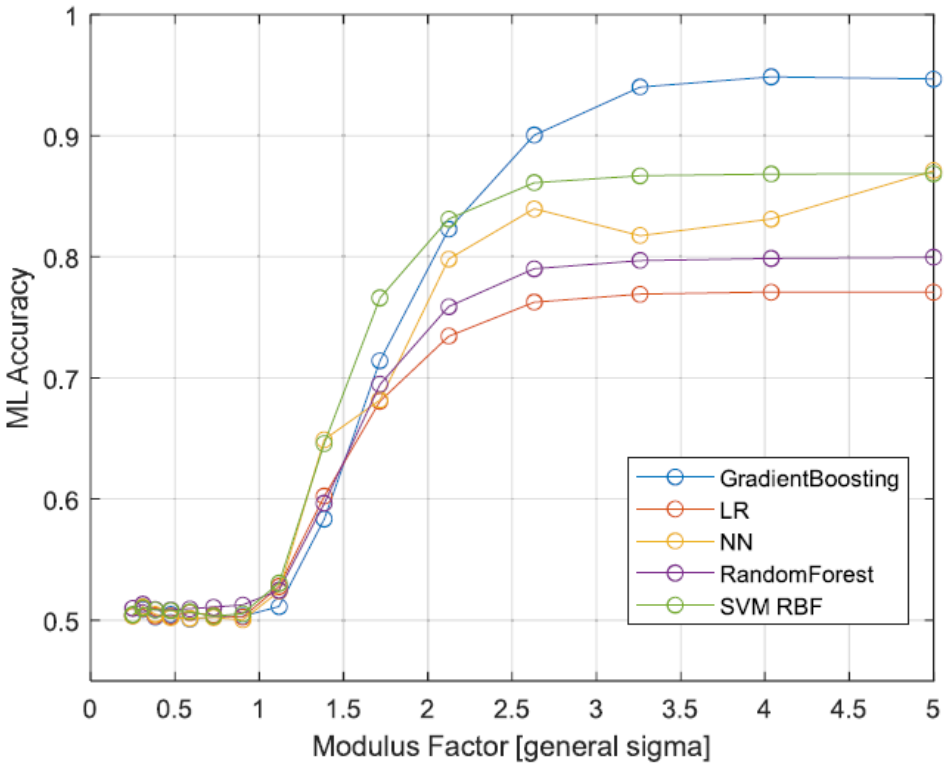


Figure 5.10: ML attack accuracy analysis on IPD-PO-PUF using single modulus factor MRO

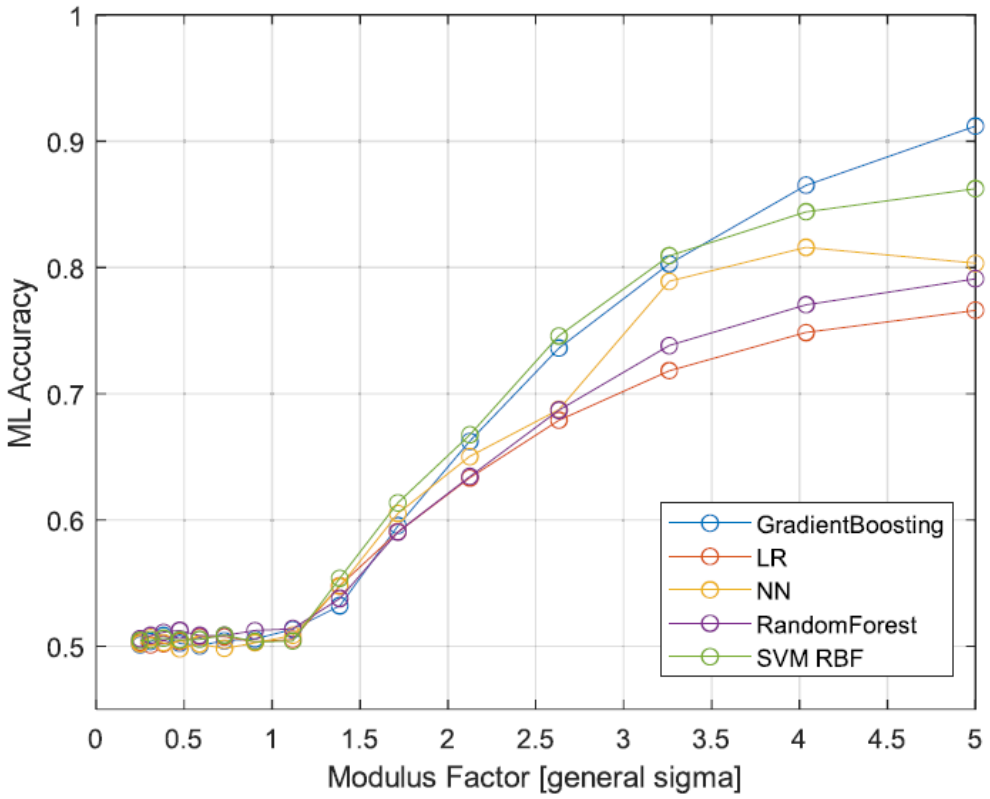


Figure 5.11: ML attack accuracy analysis on IPD-RO-PUF using AMRO with a policy of two modulus factor different by a factor of 2

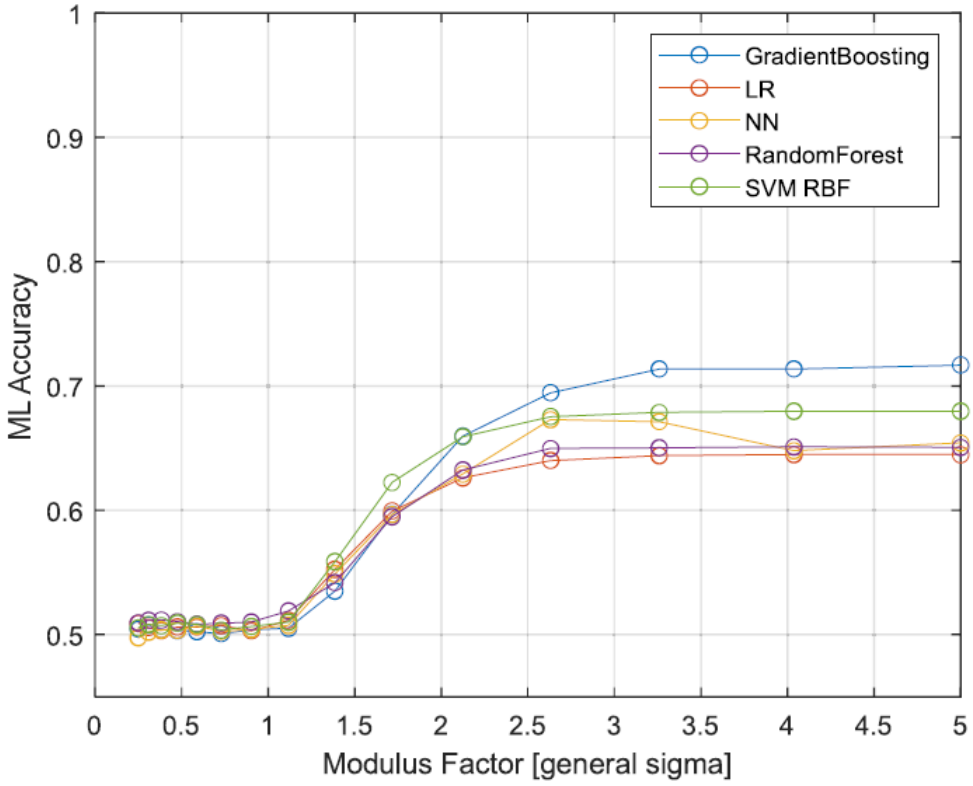


Figure 5.12: ML attack accuracy analysis on IPD-RO-PUF using AMRO with a policy of one modulus factor fixture on 1 while varying the other

5.2.2 Reliability-based Modelling

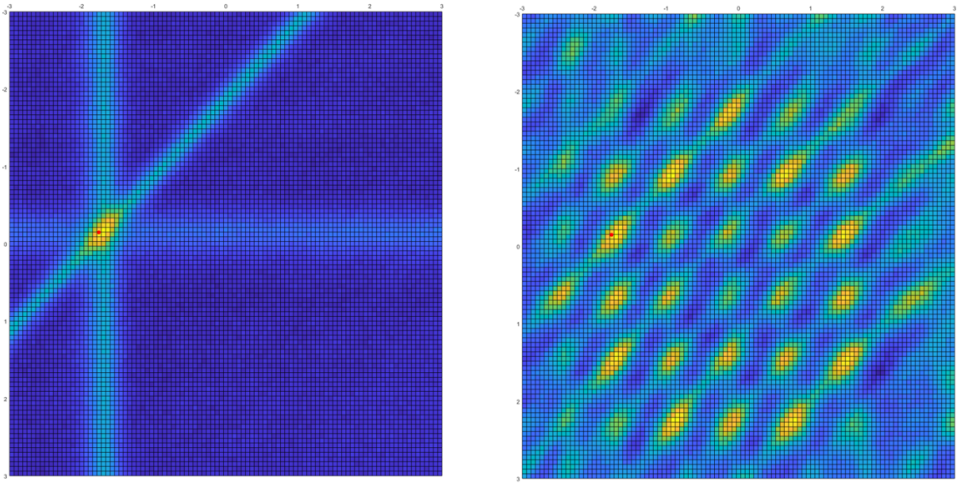
The covariance matrix adaptation evolution strategy (CMA-ES) is one of the most effective ML modeling attack methods against PUFs.[13] The idea of the reliability-based CMA-ES attack is to perform repeated measurements for the same challenge and observe its reliability performance. The greater the reliability, the farther the IRV from the decision boundary. The reliability values measured from each challenge form the target reliability vector. Similar to a traditional CMA-ES attack, the Becker's reliability attack generates several PUF models.[13] A hypothetical reliability vector is calculated from each generated PUF model.

The covariance between the hypothetical reliability vector and the target reliability vector reflects the fitness of the PUF model. The fittest PUF model, found by the CMA-ES algorithm, provides a set of modeling parameters that can best describe the target PUF. The modeling parameters can also be considered as an abstraction of the physical parameters of the PUF, such as the inverter delay of the target PUF. Some PUF designs, such as iPUF[15], are resilient to CMA-ES attacks owing to their superior reliability. However, a highly reliable PUF may be vulnerable to other CRP-based ML modeling methods.[18] Therefore, a new method to overcome such a trade-off is required.

Using MRO, we explored a different approach to improve resiliency against reliability-based modeling. To demonstrate the advantage of MRO's resilience against CMA-ES, a 40-bit PDL-RO-PUF was simulated with 1000 unique CRPs. Each unique CRP sample was sampled 100 times using an additive Gaussian noise. The CMA-ES algorithm generates parameters and evaluates their reliability and the covariance between their reliability and that of the target PUF. To better visualize the mechanism of the CMA-ES, we assumed that the CMA-ES knows 38 of the 40 delay parameters of the target PUF. A 2-D reliability covariance map can be drawn, as shown in figure 5.13. A warmer color in the covariance map represents a better correlation between the reliability vector from the modeling parameters and target reliability. The red dots in the covariance map represent the target delay values. It is evident that there is only one peak in the 2-D covariance map shown in figure 5.13 (a). The peak is also aligned with the true target delay value. The area closer to the target delay value has a higher covariance value. As there is only one peak in the 2-d map, we can extrapolate the 2-D map to a 40-D map, which would also have only one extremum.

Considering the modulus process in MRO, the number of decision boundaries increases, resulting in multiple sets of modeling parameters with similar reliability performance. By introducing multiple response decision boundaries (see figure 3.2), different points on the

x-axis can have the same reliability. As shown in figure 5.13 (b), multiple peaks are shown in the covariance map, which traps the CMA-ES algorithm in sets of parameters that have high covariance, but are different from the true delay parameter. Subsequently, the trapped parameters will yield inaccurate predictions of the responses. In the MRO designs, the number of traps increases with a decreasing modulus factor. Furthermore, the number of traps increased exponentially with the number of dimensions. For example, there are n prominent traps that have a similar covariance value to the true covariance peak in a 2D plot. Extrapolating from the 2-delay visualization to 40 delay parameter space would result in n^{20} prominent traps in the 40-D covariance space, which serves as an effective defensive mechanism against CMA-ES attacks.



(a) PDL-RO-PUF without modulus process

(b) PDL-RO-PUF with modulus process

Figure 5.13: Reliability covariance map with 2-delay parameter space

5.3 Comparison with Other PUFs

The proposed MRO-PUF demonstrates a strong potential in improving machine learning resiliency, uniqueness and uniformity while maintaining reasonably good reliability. As shown in Table 3, combining MRO and AMRO with either PDL and IPD provides close to ideal uniformity, uniqueness, and machine learning resiliency. With the modular factor configuration included in the table, the IPD has a slightly reduced reliability just above 90%. The PDL based MRO and AMRO achieved approximately 98% reliability. The performance metrics discussed have different importance in different PUF usage. [36] The proposed MRO and AMRO offers a set of tunable parameters which enables the developer to make tradeoff among the PUF properties, such as ML resiliency, reliability and area.

Table 5.5: Hardware overhead of MRO and AMRO (measured in number of LUTs)

Design		**Uniformity**	**Uniqueness**	**Reliability**	**ML Resiliency**	**Training Data Volume**	**Considered ML Method**
RS-LPUF Fine PDL[37]		51.02%	49.08%	99.49%	53.00%	200	LR, CMA-ES
Transformer PUF with 8 XOR[38]		-	49.44%	98.12%	¿ 60%	10000	LR, CMA-ES
SCA-PUF[39]		52.80%	49.90%	91%	¡60%	10000	LR, SVM-RBF, NN
CA-PUF[40]		50.06%	55.63%	92.54%	¡60%	50000	LR, SVM
XOR-mesh PUF[41]		46.04%	44.64%	92.87%	52.40%	10000	LR, SVM
64-bits 8 XOR Arbiter PUF[42]		50%	≈10%	¿98%	50.40%	24000	NN
SCROPUF[43]		49.3%	47.4%	-	-	-	-
CT-PUF[44]		50%	≈49%	¿99%	≈60%	100000	LR, SVM, NN
FF-APUF[45]		-	41.53%	97.10%	¿95%	20000	LR, CMA-ES
XOR-APUF[36]		50.73%	48.69%	99.41%	64.9%	40000	LR, SVM, NN, CMA-ES
MPUF[46]		37.03%	40.6%	-	¡80%	10000	LR, CMA-ES
PDL-RO-PUF		49.99%	73.75%	98.43%	95.74%	44447	LR, SVM-RBF, Gradient Boosting, Random Forest, NN
IPD-RO-PUF		49.31%	50.32%	98.15%	94.68%	44447	LR, SVM-RBF, Gradient Boosting, Random Forest, NN
Proposed Design	Modulus Factor						
PDL with MRO	0.6	49.80%	50.01%	97.78%	51.33%	44447	LR, SVM-RBF, Gradient Boosting, Random Forest, NN
PDL with AMRO	[0.6, 1]	49.78%	50.90%	98.09%	51.36%	44447	LR, SVM-RBF, Gradient Boosting, Random Forest, NN
IPD with MRO	0.6	49.32%	50.01%	91.46%	50.94%	44447	LR, SVM-RBF, Gradient Boosting, Random Forest, NN
IPD with AMRO	[0.6, 1]	49.34%	50.01%	93.23%	50.84%	44447	LR, SVM-RBF, Gradient Boosting, Random Forest, NN

Chapter6

Conclusion

This work proposes MRO, a novel process to improve the machine learning resiliency of PUFs using a modulus module. We begin with a discussion on the limitations of scalability and controllability in system complexity of existing RO-PUFs structures. To overcome this limitation, we proposed a modular modulus process. The proposed MRO design shows promising capability in increasing ML resiliency while preserving other PUF characteristics such as reliability. Furthermore, an alternating MRO scheme is introduced for further controllability in terms of balancing ML resiliency with reliability. We found that with the proposed scheme, the MRO design can achieve an accuracy of near-50% ML with a reliability above 90%, while maintaining or improving other performance metrics of the PUF. MRO also improves resiliency against reliability attacks by introducing a significant number of traps against the CMA-ES algorithm. This result shows the potential of MRO to improve a large variant of PUFs.

Chapter7

Future Work

The work presented in this thesis has proposed and evaluated the effectiveness of MRO and AMRO in mitigating machine learning attacks. However, we have identified challenges with the current design, particularly regarding the reliability degradation associated with aggressive modulus factors. Moving forward, we believe there are two potential directions for future exploration.

Firstly, it would be intriguing to investigate alternative entropy sources. Our experiments with PDL and IPD-RO-PUF have shown promising results in terms of improved ML resiliency. There may be other entropy sources, such as TERO-PUF, that could offer a better balance between ML resiliency and other performance metrics.

Secondly, we aim to abstract the decision boundary. One of the main goals of our proposed design is to enhance controllability in fine-tuning performance metrics. While the modulus process has been a promising method for introducing additional decision boundaries to PUF, our experiments have revealed that the decision boundary does not necessarily need to be constrained by the modulus process. For example, the discussion with the optimal uniformity is a form of redefining the decision boundary. For future research, we can explore the possibility of proposing another type of transformation that can be applied to intermediate frequency measurement, thereby introducing additional decision boundaries with more

freedom.

Bibliography

[1] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-Way Functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002. DOI: 10.1126/science.1074376. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1074376.

[2] A. Maiti, I. Kim, and P. Schaumont, "A robust physical unclonable function with enhanced challenge-response set," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1 PART 2, pp. 333–345, 2012, ISSN: 15566013. DOI: 10.1109/TIFS.2011.2165540.

[3] A. Shamsoshoara, A. Korenda, F. Afghah, and S. Zeadally, "A survey on physical unclonable function (PUF)-based security solutions for Internet of Things," *Computer Networks*, vol. 183, no. January, p. 107 593, 2020, ISSN: 13891286. DOI: 10.1016/j.comnet.2020.107593. [Online]. Available: https://doi.org/10.1016/j.comnet.2020.107593.

[4] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," *IEEE Symposium on VLSI Circuits, Digest of Technical Papers*, no. CIRCUITS SYMP. Pp. 176–179, 2004. DOI: 10.1109/vlsic.2004.1346548.

[5] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 148–160, 2002, ISSN: 15437221. DOI: 10.1145/586131.586132.

[6] A. Cherkaoui, L. Bossuet, and C. Marchand, "Design, Evaluation, and Optimization of Physical Unclonable Functions Based on Transient Effect Ring Oscillators," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1291–1305, 2016, ISSN: 15566013. DOI: 10.1109/TIFS.2016.2524666.

[7] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-Up SRAM state as an identifying fingerprint and source of true random numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2009, ISSN: 00189340. DOI: 10.1109/TC.2008.212.

[8] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen, and P. Tuyls, "The Butterfly PUF protecting IP on every FPGA," *2008 IEEE International Workshop on Hardware-Oriented Security and Trust, HOST*, no. 71369, pp. 67–70, 2008. DOI: 10.1109/HST.2008.4559053.

[9] 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, ISSN: 10816011. DOI: 10.1109/SP.2013.27.

[10] U. Ruhrmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Model building attacks on Physically Unclonable Functions," *Acmccs 2010*, vol. 2010, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1866335.

[11] Y. Ikezaki, Y. Nozaki, and M. Yoshikawa, "Deep learning attack for physical unclonable function," *2016 IEEE 5th Global Conference on Consumer Electronics, GCCE 2016*, pp. 1–2, 2016. DOI: 10.1109/GCCE.2016.7800478.

[12] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu, "Machine learning resistant strong PUF: Possible or a pipe dream?" *Proceedings of the 2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016*, pp. 19–24, 2016. DOI: 10.1109/HST.2016.7495550.

[13] 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, no. September 2015, pp. 535–555, 2015, ISSN: 16113349. DOI: 10.1007/978-3-662-48324-4{_}27.

[14] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," *Proceedings - Design Automation Conference*, pp. 9–14, 2007, ISSN: 0738100X. DOI: 10.1109/DAC.2007.375043.

[15] P. H. Nguyen, D. P. Sahoo, C. Jin, K. Mahmood, U. Rührmair, and M. Van Dijk, "The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 0, no. 0, pp. 243–290, 2019. DOI: 10.46586/tches.v2019.i4.243-290.

[16] S. V. Sandeep Avvaru and K. K. Parhi, "Feed-forward XOR PUFs: Reliability and attack-resistance analysis," *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI*, pp. 287–290, 2019. DOI: 10.1145/3299874.3318019.

[17] U. Ruhrmair, J. Solter, F. Sehnke, *et al.*, "PUF modeling attacks on simulated and silicon data," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1876–1891, 2013, ISSN: 15566013. DOI: 10.1109/TIFS2013.2279798.

[18] A. Wang, W. Tan, Y. Wen, and Y. Lao, "NoPUF: A Novel PUF Design Framework Toward Modeling Attack Resistant PUFs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 6, pp. 2508–2521, 2021, ISSN: 15580806. DOI: 10.1109/TCSI.2021.3067319.

[19] K. Stangherlin, Z. Wu, H. Patel, and M. Sachdev, "Enhancing strong puf security with nonmonotonic response quantization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 1, pp. 55–64, 2023. DOI: 10.1109/TVLSI.2022.3212271.

[20] J. Liu, Y. Zhao, Y. Zhu, C.-H. Chan, and R. P. Martins, "A weak puf-assisted strong puf with inherent immunity to modeling attacks and ultra-low ber," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 12, pp. 4898–4907, 2022. DOI: 10.1109/TCSI.2022.3206214.

[21] 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. DOI: 10.1109/WIFS.2010.5711471.

[22] B. Habib, K. Gaj, and J. P. Kaps, "FPGA PUF based on programmable LUT delays," *Proceedings - 16th Euromicro Conference on Digital System Design, DSD 2013*, pp. 697–704, 2013. DOI: 10.1109/DSD.2013.79.

[23] L. Feiten, J. Oesterle, T. Martin, M. Sauer, and B. Becker, "Systemic Frequency Biases in Ring Oscillator PUFs on FPGAs," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 174–185, 2016, ISSN: 23327766. DOI: 10.1109/TMSCS.2016.2598739.

[24] Y. Hu, Y. Jiang, and W. Wang, "Compact FPGA Ring Oscillator Physical Unclonable Functions," *TechRxiv*, 2021. DOI: 10.36227/techrxiv.14214401.v1. [Online]. Available: https://www.techrxiv.org/articles/preprint/Compact_FPGA_Ring_Oscillator_Physical_Unclonable_Functions_Circuits_Based_on_Intertwined_Programmable_Delay_Paths/14214401.

[25] L. Feiten, M. Sauer, and B. Becker, "On Metrics to Quantify the Inter-Device Uniqueness of Physically Unclonable Functions," pp. 1–11, 2016.

[26] F. Wilde, B. M. Gammel, and M. Pehl, "Spatial Correlation Analysis on Physical Unclonable Functions," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 6, pp. 1468–1480, 2018, ISSN: 15566013. DOI: 10.1109/TIFS.2018.2791341.

[27] D. P. Sahoo, D. Mukhopadhyay, R. S. Chakraborty, and P. H. Nguyen, "A Multiplexer-Based Arbiter PUF Composition with Enhanced Reliability and Security," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 403–417, 2018, ISSN: 00189340. DOI: 10.1109/TC.2017.2749226.

[28] J. Berkson, "Application of the logistic function to bio-assay," *Journal of the American Statistical Association*, vol. 39, no. 227, pp. 357–365, 1944. DOI: 10.1080/01621459.1944.10500699. eprint: https://doi.org/10.1080/01621459.1944.10500699. [Online]. Available: https://doi.org/10.1080/01621459.1944.10500699.

[29] 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. DOI: 10.1109/WIFS.2012.6412622.

[30] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[31] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston, "Random forest: A classification and regression tool for compound classification and qsar modeling," *Journal of chemical information and computer sciences*, vol. 43, no. 6, pp. 1947–1958, 2003.

[32] Y. Tanaka, S. Bian, M. Hiromoto, and T. Sato, "Coin Flipping PUF: A Novel PUF With Improved Resistance Against Machine Learning Attacks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 5, pp. 602–606, 2018, ISSN: 15497747. DOI: 10.1109/TCSII.2018.2821267.

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

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] F. Chollet *et al.* "Keras." (2015), [Online]. Available: `https://github.com/fchollet/keras`.

[36] N. N. Anandakumar, M. S. Hashmi, and M. A. Chaudhary, "Implementation of efficient xor arbiter puf on fpga with enhanced uniqueness and security," *IEEE Access*, vol. 10, pp. 129 832–129 842, 2022. DOI: `10.1109/ACCESS.2022.3228635`.

[37] N. N. Anandakumar, M. S. Hashmi, and S. K. Sanadhya, "Compact Implementations of FPGA-based PUFs with Enhanced Performance," *Proceedings - 2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems, VLSID 2017*, pp. 161–166, 2017. DOI: `10.1109/VLSID.2017.7`.

[38] Z. Wei, Y. Cui, Y. Chen, C. Wang, C. Gu, and W. Liu, "Transformer puf : A highly flexible configurable ro puf based on fpga," in *2020 IEEE Workshop on Signal Processing Systems (SiPS)*, 2020, pp. 1–6. DOI: `10.1109/SiPS50750.2020.9195259`.

[39] H. Zhuang, X. Xi, N. Sun, and M. Orshansky, "A strong subthreshold current array puf resilient to machine learning attacks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 1, pp. 135–144, 2020. DOI: `10.1109/TCSI.2019.2945247`.

[40] H. Nassar, L. Bauer, and J. Henkel, "Capuf: Cascaded puf structure for machine learning resiliency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4349–4360, 2022. DOI: `10.1109/TCAD.2022.3197539`.

[41] A. Rajan and S. Sankaran, "Lightweight and attack-resilient puf for internet of things," in *2020 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*, 2020, pp. 139–142. DOI: `10.1109/iSES50453.2020.00039`.

[42] K. T. Mursi, Y. Zhuang, M. S. Alkatheiri, and A. O. Aseeri, "Extensive examination of xor arbiter pufs as security primitives for resource-constrained iot devices," in *2019 17th International Conference on Privacy, Security and Trust (PST)*, 2019, pp. 1–9. DOI: `10.1109/PST47121.2019.8949070`.

[43] M. Choudhury, N. Pundir, M. Niamat, and M. Mustapa, "Analysis of a novel stage configurable ropuf design," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 942–945. DOI: `10.1109/MWSCAS.2017.8053080`.

[44] J. Zhang, C. Shen, Z. Guo, Q. Wu, and W. Chang, "Ct puf: Configurable tristate puf against machine learning attacks for iot security," *IEEE Internet of Things Journal*, vol. 9, no. 16, pp. 14 452–14 462, 2022. DOI: `10.1109/JIOT.2021.3090475`.

[45] C. Gu, W. Liu, Y. Cui, N. Hanley, M. O'Neill, and F. Lombardi, "A flip-flop based arbiter physical unclonable function (apuf) design with high entropy and uniqueness for fpga implementation," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1853–1866, 2021. DOI: 10.1109/TETC.2019.2935465.

[46] Q. Ma, C. Gu, N. Hanley, C. Wang, W. Liu, and M. O'Neill, "A machine learning attack resistant multi-puf design on fpga," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 97–104. DOI: 10.1109/ASPDAC.2018.8297289.