Machine Learning Attacks on Physical Unclonable Functions

A dissertation submitted in partial fulfilment of

the requirements for the degree of

BACHELOR OF *SCIENCE* in Computer Science

in

The Queen's University of Belfast

by

Oisin Kelly

24/05/2025

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Oisin Kelly Student Number: 40333616
Project Title: Machine Learning Attacks on Physical Unclonable Functions

Supervisor: Dr Chongyan Gu

Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

   By submitting your dissertation you declare that you have completed the tutorial on plagiarism at http://www.qub.ac.uk/cite2write/introduction5.html and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.
6. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students. (Please delete this if you do not agree.)

Student's signature Oisin Kelly *Date of submission* 28/04/2025

Acknowledgements

I would like to thank Dr. Chongyan Gu for her amazing assistance throughout the project, providing valuable continued feedback and suggestions on things I could add to the project to help better understand PUFs and other factors that affect them. Her continued supervision played a crucial role in shaping the direction of the project and opened my eyes up to the world of hardware security, a field I have never delved into before.

Abstract

The world of Internet of Things (IoT) is growing exponentially, and the ever growing need for lightweight security solutions is causing for a growth of primitive security devices. One proposed solution to this is the Physical Unclonable Function. Physical Unclonable Functions (PUFs) provide cost effective security by extracting unique device-specific fingerprints from manufacturing variations. However, due to the linearity of the PUFs, they are susceptible to machine learning attacks, where an attacker creates a clone of the PUF and can copy its behaviour. This project evaluates the vulnerability of different types of Strong PUFs against these modeling attacks.

Table of Contents

1.0 Introduction

1.1 Motivation

As modern computing systems become increasingly interconnected and embedded in physical environments, ensuring security at the hardware layer, especially within integrated circuits (ICs), is becoming increasingly important every day. Physical Unclonable Functions (PUFs) offer a promising alternative to traditional cryptographic key storage by leveraging intrinsic manufacturing variations in silicon ICs to produce device-unique responses. These naturally occurring inconsistencies make PUFs attractive for secure key generation and authentication. Unfortunately, physical attacks such as side-channel attacks [**1**], and more recently the rise in machine learning (ML), has allowed attackers to model these functions with increasing accuracy, raising concerns about the resilience of widely adopted PUF designs. This described challenge was one motivation that led to this project, growing the need to assess and improve the resilience of widely adopted PUF architectures.

1.2 Challenges

Basic Arbiter PUFs (APUFs) are inherently linear in their design, making them especially vulnerable to machine learning models like Logistic Regression (LR), which can learn their internal delay-based behaviour with high accuracy. More advanced designs such as XOR-Arbiter PUF (XOR-APUF) and Interpose PUF (iPUF), while increased in non-linearity and structural complexity, can be attacked by means of deeper models such as multilayer perceptrons (MLPs) [**1**] and methods like divide-and-conquer. This proposes a significant challenge in evaluating the true robustness of these PUF architectures, particularly when they are attacked under realistic conditions including noise and limited Challenge-Response-Pair (CRP) availability. Furthermore, it requires the deployment of a well-planned experimental setup that can differentiate between the limitations of existing modeling methodologies and architectural security.

In real-world environments, PUFs do not operate under ideal conditions. Environmental factors such as temperature variation, voltage fluctuation and aging introduce instability. This introduces instability in the form of noisy responses, even the same the challenged is processed repeatedly. Simulating this behaviour is a critical challenge, as the presence of noise significantly alters both the response reliability and the success of machine learning attacks. The difficulty lies in determining how to inject controlled noise into CRP datasets that simulate real-world behaviour. Moreover, incorporating countermeasures such as majority voting to mitigate noise introduces further computational overhead, requiring a repeated evaluation of the same challenge to derive an

estimated, stable response. Balancing the trade-off between realistic noise simulation and computational efficiency is a fundamental challenge addressed in this project.

Unlike basic Strong PUFs (APUFs, XOR-APUFs), the Interpose PUF (iPUF) consists of two XOR-APUFs where the response from the upper layer is interposed into the challenge of the lower layer, at a fixed position. Attacking such a structure requires a divide-and-conquer strategy, wherein the lower layer XOR-APUF must be modelled first to derive a model for the upper layer [**3**]. Designing, simulating, and attacking the iPUF accurately is significantly more complex than a standard Strong PUF which can be typically attacked directly by collecting enough CRPs by means of direct physical access or eavesdropping [**4**], and applying a ML model directly to obtain a model of the attacked PUF. A significant challenge in modeling the iPUF lies in the fact that the upper layer's response cannot be accessed directly as it is immediately embedded into the lower layer's challenge. Therefore, the attacker cannot build a conventional CRP dataset for the upper layer and must rely on successfully achieving the interposed bit through modeling. This lack of observability both complicates the simulation and attack process, making the iPUF substantially more difficult to attack by machine-learning means.

1.3 Objectives

The objectives of this project are outlined as follows:

- Develop functional and realistic simulations of Arbiter PUFs, XOR-APUFs, and Interpose PUFs, including configurable challenge lengths, XOR stream counts, and noise levels.
- Generate and evaluate large CRP datasets with added noise to simulate real-world instability.
- Implement machine learning models (Logistic Regression and Multilayer Perceptrons) capable of learning and predicting the behaviour of these PUFs.
- Assess the success rates of attacks in both noise-free and noisy conditions, evaluating the robustness of the models.
- Conduct a divide-and-conquer modeling attack strategy on the Interpose PUF and assess its effectiveness on each PUF layer individually and collectively.

1.4 Contributions

This project describes successful modeling attacks on several well-known candidates for Strong PUFs, including Arbiter PUFs, XOR-Arbiter PUFs and Interpose PUFs. The attacks work up to a given number of challenge bits (for APUF) and stream-counts (for XOR-APUF and iPUF). The prediction rates achieved by the machine learned models demonstrate the capability to accurately clone the behaviour of these PUFs.

2.0 System Requirements and Specification

2.1 Background: Compositions of Arbiter PUFs

Arbiter PUFs (APUFs) are the most extensively researched PUFs. APUFs are a straightforward delay-based PUF comprising of a sequence of n multiplexer-based switch stages to generate two competing delay paths, as shown in **Figure 1**. The challenge bit (0 or 1) at each stage determines which path a signal should traverse at each stage. The name "Arbiter PUF" derives from an arbiter (usually a latch) [**5**] at the conclusion of the two arrival lanes, which compares the arrival time of the two signals, resulting in a binary response $f(c)$. That is, if the signal on the top path arrives first, the arbiter's output would be 1; otherwise, it would be 0. Consequently, due to manufacturing variations, each stage introduces a slight delay difference. These variations are captured by modeling the PUF with an additive linear delay model (ADM) [**6**] This additive linear delay model can be captured by a Logistic Regression (LR) model to learn the internal delay-based behaviour with high accuracy.

XOR-Arbiter PUFs (XOR-APUFs) are the next iteration of Strong PUFs. A k-XOR-APUF consists of k APUFs in parallel. Each APUF is also said to be a stream or component, and the result of each stream is XORed to produce a single bit response $f(c)$, as shown in **Figure 2**. The XOR-APUF improves upon the APUF by introducing non-linearity into the model, which increases complexity and therefore increases the difficulty to attack by ML. Although not possible to attack by LR, the XOR-APUF can still be attacked by means of deep learning. A multilayer perceptron has been used to model strong PUFs since 2012 [**7**]. Furthermore, attacking an XOR-APUF with many streams (n > 7) proves to be significantly more challenging, necessitating a CRP dataset with a count in the millions [**4**]**.** In this project, up to a 9-XOR-APUF was able to be attacked.

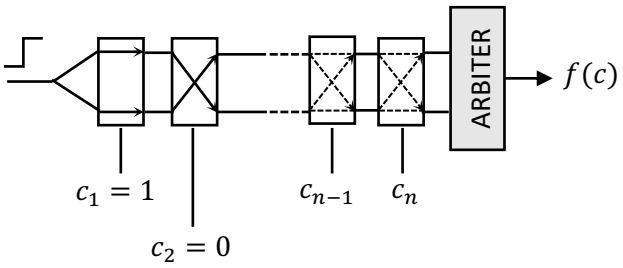

Figure 1: Schematic of n-bit Arbiter PUF with n-bit challenge c and final response as $f(c)$.

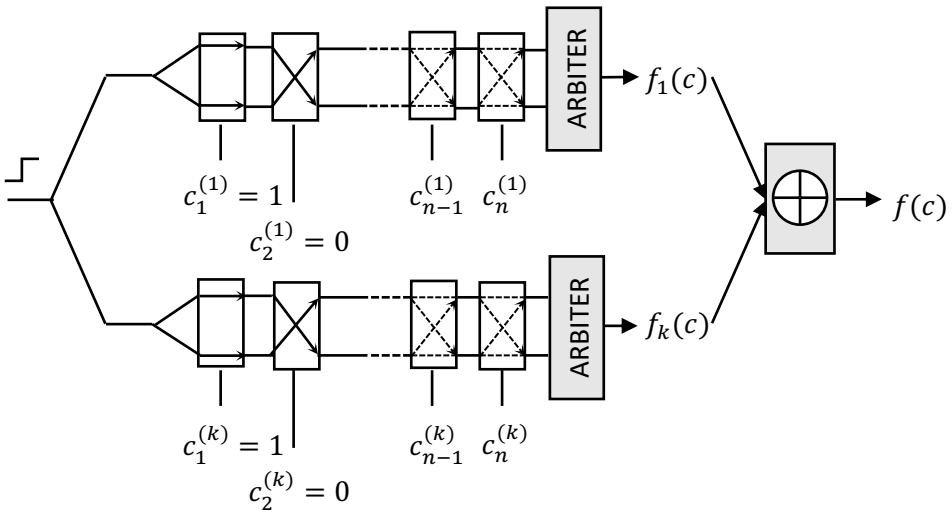

Figure 2: Schematic of 2-XOR-Arbiter PUF with n-bit challenge c and final response as $f(c)$. Individual APUF responses $f_1(c)$ and $f_k(c)$ are not visible as circuit port.

Lastly, the Interpose PUF (iPUF) [**8**] builds on top of the XOR-APUF architecture by introducing a two-layered approach to further increase complexity and non-linearity. The Interpose PUF consists of two layers, an upper layer, denoted as k_{upper}-XOR-APUF and a lower layer, denoted as k_{lower}-XOR-APUF. The upper layer consists of a n-bit XOR-APUF, and the lower layer consists of a $(n+1)$-bit XOR-APUF. The upper layer interpolates its response $f_u(c)$ into the challenge of the lower layer, as denoted by c_l, at a fixed position (e.g. the middle challenge bit), as shown in **Figure 3**. Although a high complexity iPUF cannot be directly attacked by training a MLP on its Challenge-Response-Pairs alone, a model of the upper layer can be attacked by a divide-and-conquer attack method on the iPUF [**3**]. This model can then be used to derive a sufficiently sized CRP dataset to attack the lower layer.

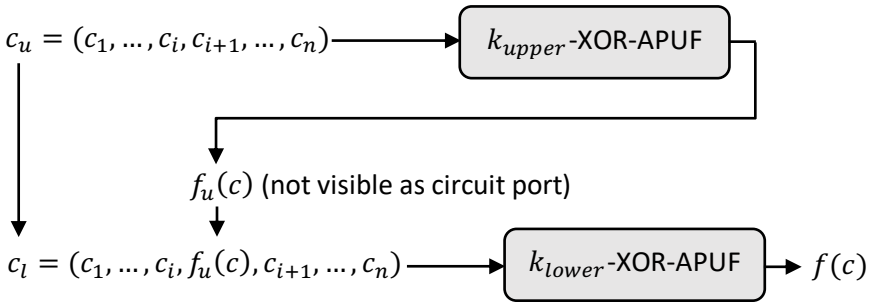

Figure 3: Schematic of n-bit (k_{upper}, k_{lower})-Interpose PUF. Upper and lower XOR-APUFs represented by k_{upper} and k_{lower} respectively. The response of the upper layer, as denoted by $f_u(c)$, is interposed in the middle bit of the lower challenge. The final response $f(c)$ is determined by k_{lower}.

2.2 Modelling Techniques

Physical Unclonable Functions can be modelled by machine learning algorithms using collected challenge-response-pair data, effectively creating a software clone of the PUF. Several studies have shown that given enough Challenge-Response Pairs (CRPs), an attacker can train a model to accurately predict a PUF's responses.

Logistic Regression (LR) is one of the earliest and most effective modeling techniques for attacking Arbiter PUFs (APUFs). This is due to the linearity of the APUF's internal delay model, which allows LR to closely approximate the mapping between challenge inputs and output responses. Since APUFs behave deterministically and follow an additive delay structure, LR can often predict responses with over 99% accuracy given a relatively small CRP dataset [**9**]. Numerous studies have demonstrated this vulnerability. Rührmair et al. [**10**] showed that LR could successfully model APUFs even with moderate length challenges (e.g. 64 bit).

To increase resistance against simple ML models like LR, researchers introduced the XOR-Arbiter PUF (XOR-APUF), which combines the outputs of multiple APUFs using an XOR gate. This significantly increases the non-linearity of the PUF's output, making it difficult for linear models to learn. Despite this added complexity, multilayer perceptrons (MLPs) have been proven capable of modeling XOR-APUFs with high accuracy, especially for lower XOR counts (e.g. up to 7 streams). MLPs can learn non-linear relationships in the data by using hidden layers with non-linear activation functions. The effectiveness of MLPs in modeling XOR-APUFs has been demonstrated in studies where a deep learning approach was able to break up to 9-XOR APUFs with high accuracy [**4**]. These findings show that XORing APUFs increases security but does not guarantee robustness against sufficiently powerful models and large enough datasets.

The Interpose PUF (iPUF) [**8**] was proposed as a more secure architecture, combining two XOR-APUFs in a layered manner. In an iPUF, the output from the upper layer (an XOR-APUF) is inserted into a fixed position of the lower layer's challenge, making the challenge-response mapping dependent on internal logic rather than just the external input. This interdependency was designed to defeat direct modeling attacks on the whole structure and to remove linearity. However, divide-and-conquer (DAC) strategies have been developed to overcome this defence. In a DAC attack, the attacker first models the lower layer. When a decently modeled lower layer has been obtained, the attacker can then model the upper layer by carefully choosing responses that are interposed into the lower and that correspond to the correct output of the iPUF as a whole. Once an accurate approximation of the upper layer is

obtained, the predicted output bit (i.e., the interposed bit) an inserted into the challenges of the lower layer. This two-stage attack highlights a critical limitation of the iPUF and structural defences as a whole: if any component of the system can be individually modelled, the entire system may still be vulnerable to modeling.

2.3 Related Work

To be completed.

2.4 Organisation of the Paper

This dissertation is structured as follows. Section 3 contains the background of the Physical Unclonable Functions (PUFs) attacked, along with the machine learning methodologies employed to attack them. Section 4 describes the implementation process, the simulation of PUFs, CRP generation, and the software libraries used. Section 5 outlines the experimental setup and evaluation strategy, including parameters used such as noise levels and number of CRPs, and the assessment metrics applied. Section 6 discusses the experimental results, analysing the performance of the machine learning attacks across different PUF designs. Section 7 concludes the dissertation with a summary of findings and suggestions for future work.

3.0 Methodology

This section outlines the theoretical design of the Physical Unclonable Functions (PUFs), and the machine learning (ML) techniques used to model them. It details the methodology behind simulating Arbiter PUFs, XOR-APUFs, and Interpose PUFs, as well as the attack strategies used to predict their responses. Noise resilience is implemented using majority voting, where the mode of multiple responses to the same challenge is taken as the final response. The resulting effective noise level can be estimated using the binomial distribution. Machine learning models, including Logistic Regression and multilayer perceptrons, are selected based on the linearity or non-linearity of each PUF, and a divide-and-conquer approach is used to model the layered iPUF structure.

3.1 PUF Models

Arbiter PUFs

Arbiter PUFs (APUFs), as described in Section 2.1, consist of a sequence of n multiplexer-based stages. Two electrical signals race simultaneously through these stages [**1**]. The signal paths within an APUF are controlled by a sequence of n external input bits, denoted as $c_1, c_2, \dots, c_n$, where each bit i-th bit is applied at the i-th stage. After the last stage, an arbiter usually consisting of a latch, determines

which of the two arrival lanes arrived first and outputs a corresponding binary output: 0 or 1. This can be seen in **Figure 1**. This sequence of input bits forms the PUF's challenge $c = c_1, c_2, \dots, c_n$, and the output of the arbiter is referred to as the response r. The value n is often referred to as the bitlength of the PUF.

It has become standard to describe the functionality of APUFs via an additive linear delay model [**10**]. For each n-bit challenge $c \in \{0,1\}^n$, it can be proved than the response of an APUF for a given challenge c is given by:

$$r = \begin{cases} 1, & \text{if } \Delta_c < 0, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

and the delay difference at the arbiter, Δ_c is given by: $\Delta_c = \vec{w}^T \Phi$, where $\vec{w}$ is known as the $weight\ vector$, and is a $(n+1)$-dimensional vector of real numbers, whose components depend on the path delay. Φ is the parity vector derivable from the given challenge c, whose components are given by

$$\Phi_n = 1, \text{ and } \Phi_i = \prod_{j=i}^{n-1} (1 - 2c_j), i = 0, 1, \dots n-1\,. \quad (2)$$

Therefore, the success of modeling an APUF relies on accurately predicting the weight vector $\vec{w}$ [**5**]. Modelling an APUF can be achieved quite successfully by several machine learning models, such as Support Vector Machine (SVM) [**6**], and in this paper using Logistic Regression (LR).

XOR-Arbiter PUFs

To enhance the resilience of the APUF against machine learning-based attacks, it was proposed to combine the outputs of multiple APUFs using the XOR operation with Each APUF having the same bitlength [**11**]. This construction, known as an k-XOR-APUF, is depicted in **Figure 2** for $k = 2$. Increasing the number of XORed APUFs introduces non-linearity into the overall challenge-response behaviour, effectively obscuring the underlying delay relationships that simpler machine learning models attempt to capture. While this significantly improves the PUF's modeling resistance, it also increases the complexity of the resulting response function, which in turn increases the number of Challenge-Response-Pairs (CRPs) and model parameters needed to achieve a successful modeling attack such as multilayer perceptrons [**1**]. The mathematical model for the k-XOR-APUF is shown as:

$$response_{XOR} = \prod_{i=1}^{k} \operatorname{sgn}\left(\vec{w}_i^T \Phi_i\right) = \operatorname{sgn}\left(\prod_{i=1}^{k} \vec{w}_i^T \Phi_i\right) \quad (3)$$

where Φ is the parity vector that corresponds to the applied challenge vector [**5**].

Interpose PUFs

With the knowledge that the XOR-APUF can be attacked relatively easily with a DNN and sufficient CRPs, the Interpose PUF was proposed to defeat ML models that heavily rely on CRP-based attacks. A (k_{upper}, k_{lower})-iPUF consists of two layers. The upper layer consists of an n-bit k_{upper}-XOR-APUF and the lower layer consists of an $(n+1)$-bit k_{lower}-XOR-APUF. We denote the input to the k_{upper}-XOR-APUF as $c_u = (c_1, \dots, c_i, c_{i+1}, \dots, c_n)$. The response $f_u(c)$ of the k_{upper}-XOR-APUF is interposed into c_u at a fixed position to create a new $(n+1)$-bit challenge $c_l = (c_1, \dots, c_i, f_u(c), c_{i+1}, \dots, c_n)$ [**8**]. This structure is shown in **Figure 3**. This two-layered structure completely removes all linearity from the additive linear delay model. A splitting attack method will need to be conducted to gather information on the lower layer's behaviour before any modeling can be conducted on the upper layer.

3.2 Majority Voting

PUFs inherently exhibit noisy behaviour due to environmental variations like voltage instability, temperature fluctuations, and device aging. To enhance the reliability of noisy responses in a repeatable and quantifiable manner, a common design technique is majority voting [**12**]. In this approach, a given challenge is applied to the PUF multiple times, denoted by an odd number n. The final response is determined by calculating the mode of the n responses. This technique reduces the effect of noise-induced response errors by selecting the most observed output across repeated evaluations, thereby improving the overall reliability and consistency of the generated CRPs.

The probability that the majority vote produces an incorrect response, denoted as P_{error}, can be modelled using the binomial distribution. For a per-response error rate p, the probability that more than half of the repeated responses are incorrect is given by:

$$P_{error} = \sum_{k=\frac{n+1}{2}}^{n} \binom{n}{k} p^k (1-p)^{n-k} \tag{4}$$

This expression estimates the chance that enough incorrect responses occur to outweigh the correct ones, leading to an inaccurate result. By reducing the effect of random noise, majority voting improves the reliability of PUF response data. This in turn supports the development of more accurate and robust machine learning models trained on noisy CRP datasets.

3.3 Employed Machine Learning Models

Logistic Regression

Logistic Regression (LR) is a well investigated supervised machine learning framework [**13**] that establishes a linear probabilistic model between input variables $x$and a binary output variable y by constructing a probabilistic model. LR has previously proven effective in modeling linearly additive delay-based PUF architectures, such as the Arbiter PUF (APUF) [**1**]**.** In this work, the LR model was implemented using the Scikit-learn Python library [**14**], which employs the LBFGS [**15**] solver by default for efficient optimisation and training of the model parameters. We assume that the hypothetical function is [**16**]:

$$h_\theta(x) = g(\theta^T x), g(f) = \frac{1}{1+e^f}, \tag{5}$$

where g is a sigmoid function used by the LR for classification, x is the input, and θ is the parameter to be solved in LR. Then the decision boundary is:

$$f = \theta^T x = 0. \tag{6}$$

The LR determines a decision boundary and uses it to predict the probability that a given input belongs to a particular binary class. This method assumes that the response data follow a Bernoulli distribution. By maximising a likelihood function, gradient-based optimisation techniques are applied to estimate the model parameters, represented by θ. The sigmoid function transforms the decision boundary into probabilities, resulting in a binary classification decision: if $\theta^T x \geq 0$, the prediction is $y = 1$; otherwise, $y = 0$. LR has been successfully applied to model APUFs and XOR-APUFs since as early as 2010 [**1**]. Given that PUF responses are inherently binary, modeling attacks using LR naturally become binary classification problems. The specific decision boundary varies depending on the type of PUF. For the APUF, according to [**1**], the decision boundary is expressed as [**16**]:

$$f = \vec{w}^T \vec{\Phi} = 0, \tag{7}$$

where $\vec{w}$ is the weight vector of the APUF and $\vec{\Phi}$ is the parity vector derived from the challenge bits, as previously described in Eq. (2).

Multilayer Perceptron

In this project, two different architectures of Multilayer Perceptrons (MLPs) were employed; one better suited for attacking XOR-APUFs and the other for iPUFs. The first MLP that explained offers better modeling towards XOR-APUFs:

MLP for XOR-Arbiter PUF

Neural network methods, as demonstrated in [**4**], are more effective in attacking PUFs in terms of performance and training data size compared to LR and SVM [**1**, **18**]. This study motivated the development of a deep neural network for attacking large XPUFs. Deep neural networks, characterised by their larger number of processing layers, are particularly effective in complex classification problems [**17**]. However, selecting an appropriate number of layers and neurons in each hidden layer remains a challenge for many problems.

Since an XOR-APUF contains multiple APUFs, each additional APUF makes the relationship between the challenge and response more complicated. Specifically, a k-XOR-APUF divides the space of input challenges into multiple sections using k boundaries, known as hyperplanes. These hyperplanes separate the challenge space into regions where each region has either a response of 0 or 1, and neighbouring regions having different values for responses [**4**]. Even though the output is binary, having a large number of APUFs creates many regions, possibly up to 2^k. This results in a complex, highly variable response pattern. Because complex patterns are difficult for simpler methods such as LR (which can be thought of as a single-layer NN), DNNs with multiple layers and neurons are usually better suited for modeling XOR-APUFs [**19**].

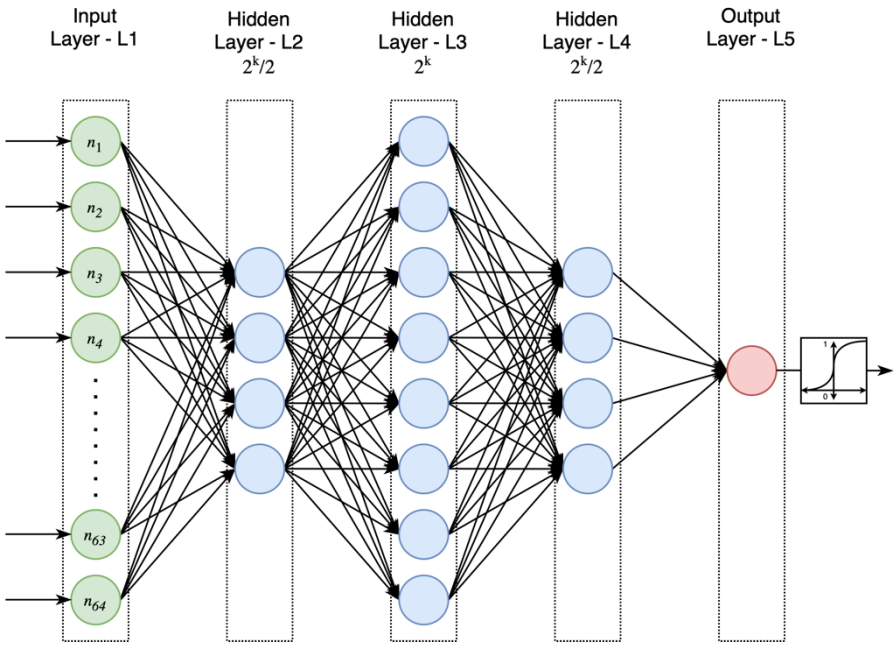

Figure 4: MLP architecture proposed by [**4**]. The MLP architecture has five layers, among which there are three hidden layers, and the activation function of the hidden layers is tahn. The input of the input layer is the parity vector corresponding to the challenge, and the activation function of the output layer is sigmoid.

Figure 4 illustrates the Multilayer Perceptron architecture adapted from the design proposed by Mursi et al. [**4**]. The model consists of five layers: an input layer, three hidden layers, and an output

layer. The input to the network is the parity vector derived from the PUF challenge. The hidden layers are fully connected, with the number of neurons in each layer determined as follows: the first and third hidden layers each contain $\frac{2^k}{2}$ neurons, while the second contains 2^k neurons. Each hidden layer uses the hyperbolic tangent activation function:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{8}$$

This activation function was selected based on the findings in [**4**], which demonstrate that $\tanh$ is particularly well-suited to approximating the binary-valued decision boundaries produced by complex PUF structures such as XOR-APUFs. In contrast to Rectified Linear Unit (ReLU), which may require deeper networks to achieve similar performance, the $\tanh$-based architecture showed improved modeling accuracy with fewer layers. The output layer uses the sigmoid activation function to produce a probability in the range $[0, 1]$, allowing for binary classification of the PUF response. To optimise the model during training, the *Adam* optimiser [**20**] was used due to its efficiency and adaptive learning rate properties. The Binary Cross-Entropy (BCE) loss function was applied to evaluate prediction error during training. The BCE loss function is defined as [**5**]:

$$L_{\text{BCE}}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})] \tag{9}$$

where y_i and $\hat{y}_i$ correspond to the original responses and predicted responses of the PUF, respectively, and n is the number of responses considered. This architecture, combining $\tanh$ activation, sigmoid output, Adam optimisation, and binary cross-entropy loss, balanced modeling accuracy with computational efficiency and was effective for modeling XOR-APUFs with up to 9 streams.

MLP for Interpose PUF

For modeling the more structurally complex Interpose PUFs (iPUFs), a different deep learning strategy was employed, adapted from [**5**]. While the XOR-APUF attacks leveraged the hyperbolic tangent ($\tanh$) activation due to the relatively simpler classification boundaries, the DNN designed for iPUF modeling adopted the Rectified Linear Unit (ReLU) activation function in its hidden layers and input layer:

$$\text{ReLU}(x) = \max(0, x). \tag{10}$$

ReLU was chosen because it allows deeper neural networks to train more easily without running into problems like vanishing gradients [**21**]. This was important for modeling the highly non-linear and layered challenge-response behaviour of iPUFs. Using ReLU also helped the NN learn the complex decision boundaries created by the structure of the iPUF, although it required a deeper network compared to models using the $\tanh$ activation. Training of the model used the Adam optimizer and

Binary Cross-Entropy loss function, following the same method as for the previous model. Overall, the use of ReLU and a deeper architecture was necessary to model the more complicated patterns in the iPUF responses, where multiple XOR operations and interposed challenges make modeling significantly harder. **Figure 5** represents the architecture of the NN:

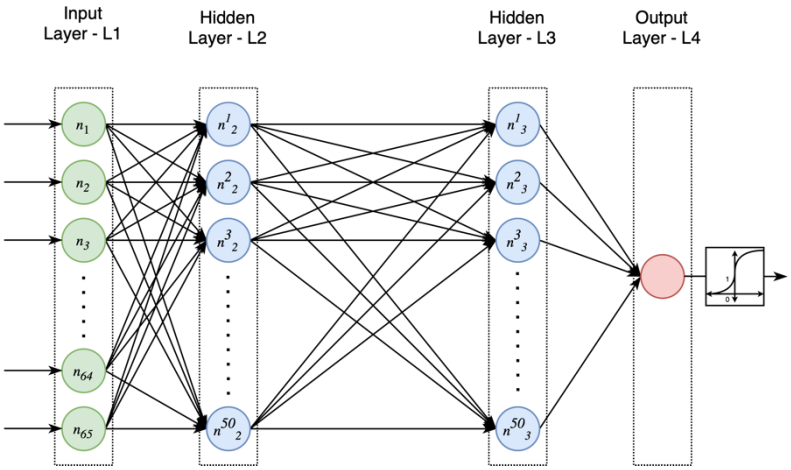

Figure 5: MLP architecture adapted from [**5**] for modeling iPUFs. The layers use a ReLU activation instead of tanh. The two middle layers each contain 50 neurons. The input of the input layer is the parity vector corresponding to the challenge, and the activation function of the output layer is sigmoid.

Divide-and-Conquer Attack on Interpose PUFs

The Interpose PUF (iPUF) introduces structural complexity by immediately interposing the response of an upper layer XOR-APUF into the challenge of a lower layer XOR-APUF, disabling any form of side-channel attack on the upper layer's responses, therefore obstructing straightforward CRP-based modeling attacks. This interdependency between layers significantly reduces the efficacy of standard supervised learning methods. To overcome this obstacle, a divide-and-conquer (DAC) attack strategy was proposed in [**3**], where the two-layered structure is decoupled into two sequential modeling phases.

Contrary to initial assumptions, the attacker first attempts to model the lower layer by constructing a synthetic training set. This is achieved by recording a CRP dataset for the iPUF, taking the challenges and inserting randomly guessed bits at the interposition position (typically the middle challenge bit). Crucially, this approach works because for 50% of cases, the interposed bit has no effect on the final output response [**3**], and for the remaining 50% of cases, a random guess of the interposed bit is correct half the time. This yields an effective accuracy rate of ≈75% for the lower layer's behaviour. With a suitably sized CRP dataset of the iPUF, learning methods such as MLP can achieve a validation accuracy rate of >90% for some iPUF configurations despite this noise level. This is crucial, as this accuracy rate surpasses the estimated accuracy of the training set (≈75%).

Having obtained a model of the lower layer XOR-APUF, the attacker proceeds to the second phase targeting the upper layer. By applying the lower-layer model to predict outputs for both possible bit values (0 and 1) at the interposition position, the attacker can infer the correct interposed bit by comparing the iPUF's actual response against these predictions. Specifically, when the two predicted responses differ (indicating the interposed bit affects the output), the attacker selects the bit value whose prediction equals the observed iPUF response. This procedure is described in **Algorithm 1** [**3**]:

Algorithm 1 Heuristic for creating upper-layer CRP training set

procedure HEURISTIC($C, R, \hat{f}_d$)
    initialise empty training set (C_U, R_U)
    for c, r in C, R **do**
        $c^{(1)} \leftarrow (c_1, \dots, c_{n/2}, 1, c_{n/2+1}, \dots, c_n)$
        $c^{(0)} \leftarrow (c_1, \dots, c_{n/2}, 0, c_{n/2+1}, \dots, c_n)$
        if $\hat{f}_d(c^{(1)}) = \hat{f}_d(c^{(0)})$ **then**
            continue
        end if
        if $\hat{f}_d(c^{(1)}) = r$ **then**
            $add\ (c, 1)\ to\ (C_U, R_U)$
        else
            $add\ (c, 0)\ to\ (C_U, R_U)$
        end if
    end for
    return (C_U, R_U)
end procedure

Algorithm 1: Heuristic for creating Challenge-Response-Pair training set for the upper layer when provided a model with decent accuracy for the lower layer, represented as $\hat{f}_d$, and a training set for the complete iPUF, represented as (C, R). The algorithm identifies challenges where the interposed bit affects the output, then infers the upper layer's response by comparing the lower layer model's predictions for both possible bit values (1 or 0) against the true response r. Challenges that do not teach anything about the upper layer's behaviour are discarded, and others are retained with the inferred interposed bit.

Even with the upper layer modelled, only 50% of responses from the upper layer do not influence the response of the lower layer. With a substantially high accuracy model achieved of the lower layer by interposing random bits, it can be assumed that the attack can be concluded after the first phase where the iPUF is attacked with an accuracy of >90% without modeling the upper layer, with the iPUF acting as a single XOR-APUF (lower layer). However, creating a model of the upper layer further increases accuracy of a modelled iPUF, and is necessary for the modeling of a complete Interpose PUF.

The entire Interpose PUF can be attacked using an iterative divide-and-conquer (DAC) strategy. This strategy involves breaking down the modeling of the PUF into two interleaved phases that mutually refine each other. Initially, a model of the lower layer is trained using synthetic data generated by inserting randomly guessed interposed bits into the challenge. This model is then used to infer training data for the upper layer by determining which bit value at the interposition position best matches the PUF's actual response. Subsequently, a model of the upper layer is trained using this inferred data, and its predictions are used to update the interposed bits in the lower layer's challenge, thereby producing a more accurate dataset for retraining the lower-layer model. This iterative process continues, with each model progressively improving the quality of the training data for the other, until

the overall prediction accuracy of the iPUF model converges above a predefined threshold. The full DAC procedure is outlined in **Algorithm 2** [**3**]:

Algorithm 2 Divide-and-Conquer Interpose PUF Attack

procedure ATTACK(n, C, R)
  $C_d \leftarrow$ INTERPOSE$(C,$ random bits$)$ # Create training set for lower layer
  $\hat{f}_d \leftarrow \mathrm{MLP}_{n+1}(C_d, R)$ # Train lower layer
  while test accuracy is below target **do**
    $C_u, R_u \leftarrow$ HEURISTIC$\left(C, R, \hat{f}_d\right)$ # Create training set for upper layer
    $\hat{f}_u \leftarrow \mathrm{MLP}_n(C_u, R_u)$ # (Re-)train upper layer
    $C_d \leftarrow$ INTERPOSE $\left(C, \hat{f}_u(C)\right)$ # Create training set for lower layer
    $\hat{f}_d \leftarrow \mathrm{MLP}_{n+1}(C_d, R)$ # Re-train lower layer
  end while
  return $\hat{f} : c \mapsto \hat{f}_d(c_1, \ldots, c_{n/2}, \hat{f}_u(c), c_{n/2+1}, \ldots, c_n)$ # Final Interpose PUF Model
end procedure

Algorithm 2: Divide-and-Conquer modeling attack on the Interpose PUF [**3**]. This procedure outlines an iterative attack strategy for reconstructing the response function of the full Interpose PUF. The input consists of a challenge bit-length n, and a challenge-response-pair dataset (C, R) for the complete iPUF. The attack begins by training a model of the lower layer using synthetic challenges generated by interposing random bits. With this initial model $\hat{f}_d$, the algorithm applies **Algorithm 1** to heuristically infer a training set for the upper layer. A model $\hat{f}_u$ is trained for the upper layer using this set, and its predictions are then used to refine the interposed bits for re-training the lower layer. This cycle is repeated until the test accuracy of the lower layer model reaches a target threshold. The final model $\hat{f}$ represents the iPUF's response as the output of the lower layer given a challenge interposed with a predicted bit from the upper layer model.

4.0 Implementation

4.1 Languages, Environments and Libraries

This system was implemented in Python [**22**], specifically Python 3.10. Python was chosen due to its high adoption in machine learning and security research. Python was also chosen by research papers [**4**, **5**], which reassured the decision to adopt it. Its popularity stems from its readable syntax, ease of use, and the extensive ecosystem of libraries and community-driven support.

Python was particularly well-suited to the challenge of building, simulating, and attacking Physical Unclonable Functions (PUFs) because of its seamless integration with key machine learning libraries. For example, Scikit-learn was employed for implementing Logistic Regression (LR) models. This framework provides efficient, out-of-the-box tools for supervised learning tasks and is ideal for lightweight classification models such as LR, which are well-matched to the linearly additive properties of APUFs.

For more complex and non-linear models, such as those necessary to model XOR-APUFs and Interpose PUFs, the Keras deep learning framework was used. Keras, built upon TensorFlow, was implemented independently as a high-level API. Its modular design, user-friendly syntax, and rapid prototyping capabilities made it beneficial for constructing multilayer perceptrons (MLPs) to model XOR-APUF and iPUF structures. Keras also simplifies the training and tuning of neural networks by providing built-in support for widely used optimisers (e.g., Adam) and loss functions (e.g., Binary Cross-Entropy), both of which were used in this project.

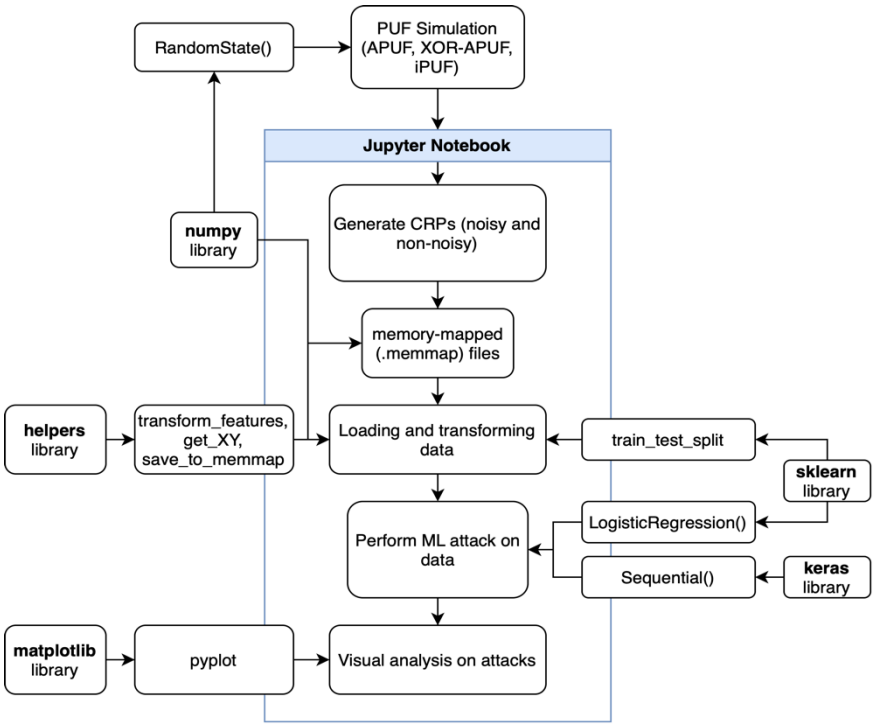

Figure 6: Overview of the implementation workflow for simulating PUF architectures and conducting machine learning-based modeling attacks within a Jupyter Notebook environment.

All modelling, data processing, and experimentation were carried out within Jupyter Notebooks, as displayed in **Figure 6**, running in Visual Studio Code (VSCode). This environment was selected for its flexibility and interactive interface, which are highly beneficial when working with large datasets and iterative model development. Jupyter notebooks are also widely adopted in machine learning research, allowing for code documentation and visualisation side-by-side. This approach played a crucial role in visualising the execution of various attacks and debugging model outputs in real-time.

For plotting and evaluation, Matplotlib was used to generate visual comparisons and performance graphs, such as visualising the accuracy of models with supplied different number of CRPs. Additionally, NumPy was used for all numerical operations, particularly in challenge

transformation such as applying a cumulative product to challenges before sending them to a ML model, CRP generation, and PUF simulation logic.

All PUF simulations were created in Python classes: APUF (ArbiterPUF), XOR-APUF (XorPUF), and Interpose PUF (InterposePUF), which abstracted the functionality for challenge input, response generation, and noise injection. To ensure reproducibility across runs and training iterations, which is paramount for machine learning analysis, NumPy's RandomState class was used for initialising the delay-weight parameters of each APUF instance.

4.2 Dataset Organisation

The generated CRPs used are generated with a clear and systematic naming convention to provide a clear indication of which files held what and ensured reproducibility. Challenges are represented by a list of 1's and 0's, and responses are represented by a single bit, 1 or 0. Each dataset is labelled with key information about the PUF, such as the challenge bit length, the number of CRPs, the percentage of noise in the dataset, and in case of the XOR-APUF CRPs, how many streams (stages) it had. This approach allows for easy identification and proper organisation. The challenge and response files are stored separately memory-mapped files (.memmap). This file format, supported by NumPy, allows for efficient access during model training, such as data slices without loading entire datasets in RAM.

Dataset file names included:

- 9 Stream XOR-APUF; 5% noise:
    - crps/xor_puf/5noise/9XOR_64bit_chal_20000000.memmap, and
    - crps/xor_puf/5noise/9XOR_64bit_resp_20000000.memmap
- (7, 7) Interpose PUF:
    - crps/interpose_puf/7_7iPUF_64bit_chal_20000000.memmap, and
    - crps/interpose_puf/7_7iPUF_64bit_resp_20000000.memmap

4.3 Custom Utility Functions and Data Handling

Mentioned in **Figure 5**, though not explained, are three utility functions that were developed as part of this project to support data preprocessing, transformation and efficient storage within the system. These functions were essential for handling large amounts of CRPs generated during the simulation and modeling of PUFs. All functions are found in the *helpers.py* file.

The *transform_features* function was used to convert raw binary challenge vectors into the feature representation required by the additive linear delay model. This function first transforms the

binary inputs from $C \in \{0, 1\}$ into bipolar format $C_{bp} \in \{-1, 1\}$ using the transformation $C = 2C - 1$. The bipolar challenge matrix is then flipped left-to-right using NumPy's *fliplr* function, and a cumulative product is applied across each row using NumPy's *cumprod* function. This transformation encodes the parity information required by the model, resulting in a feature vector Φ such that each element $\Phi_i = \prod_{j=i}^{n-1}(1 - 2c_j)$, as described in Eq. (2). An example for an 8-bit challenge is as follows:

```
Original binary challenge: [[0 1 1 0 0 1 0 0]]
Transformed challenge:     [[-1  1  1 -1  1  1  1 -1]]
```

The *get_XY* function is used to extract challenges and responses from data generated by the PUF simulations, where each entry is a list consisting of a binary challenge followed by a single response bit at the end. This format keeps the response directly coupled with its corresponding challenge, maintaining the structural integrity of the Challenge-Response-Pair. The function separates these components by removing the final element as the response and retaining the rest as the challenge. Both are returned as NumPy arrays, enabling further processing like that of *transform_features* and efficient processing in machine learning workflows. An example for a 15-bit challenge and 1-bit response is as follows:

```
Before challenge-response splitting:
[[1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1]]

After challenge-response splitting:
[[1 1 1 0 0 1 0 0 1 0 0 0 1 1 0]] [1]
```

The *save_to_memmap* function is responsible for persisting the CRP dataset NumPy arrays into memory-mapped files (.memmap). This approach allows the data to be stored on disk while still enabling efficient, array-like access during model training and evaluation. By using NumPy's *memmap* functionality, only required slices of data are loaded into memory at any given time, reducing RAM usage when working with millions of CRPs.

5.0 Experimental Setup

5.1 PUF Simulation and CRP Generation

All CRPs were generated by a simulation of each PUF. Generating CRPs on silicon would provide more realistic results, with environmental factors at play with the noise level such as temperature and voltage, however with the flexibility of running simulations at any point with a simulation of the PUF and the ability to control factors such as noise level, the decision was made to generate the CRPs synthetically.

To mitigate the impact of noise in a CRP dataset, a majority vote mechanism (see Section 3.2) was incorporated when generating the CRPs for each of the noise levels for the APUF, and for XOR-APUF with 20% noise. This was done to evaluate the efficacy of such measures when generating CRPs of a PUF. However, in a real-world scenario, this mitigation technique would only work with direct access to the PUF, as eavesdropping might not supply enough of the same challenge and corresponding response to apply the majority vote on. **Table 1** displays the final estimated noise level after applying the majority vote for each noise level with a 3 out of 5 majority votes:

Noise Level	Responses per Challenge	Majority Votes	Final Noise Level (Estimated)
5%	5	3+	0.12%
10%	5	3+	0.86%
20%	5	3+	5.79%

Table 1: Estimated final noise levels after applying a majority vote mechanism with 5 repeated responses per challenge.

Attacking Arbiter PUF

For APUF modeling attacks, attacks were conducted on a 64 and 128 bitlength APUF with varying amounts of CRPs and noise levels (see **Table 2**). Noise levels included 0%, 5%, 10% and 20%, with a majority vote mechanism incorporated for each noise level. This was done to ensure a comprehensive evaluation of the LR model's accuracy under both ideal (noise-free) and realistic (noisy) operating conditions, and to see if the LR model could see past the noise and achieve a high prediction accuracy despite the noise level. By varying the number of CRPs and noise level, it became possible to systematically investigate the minimum number of CRPs require to achieve a high prediction accuracy under different scenarios.

Arbiter PUF bitlength	CRPs used	Noise Levels
64 bits and 128 bits	100 1,000 10,000 100,000	0% noise 5% noise 5% noise with Majority Voting 10% noise 10% noise with Majority Voting 20% noise 20% noise with Majority Voting

Table 2: Challenge-Response-Pair (CRP) counts used for Arbiter PUF modeling across 64 bit and 128 bit configurations for different noise levels.

Attacking XOR-Arbiter PUF

Each k-XOR-APUF attacked had a 64 bitlength, where k is the number of streams the XOR-APUF has, and ranges from 4 to 9 inclusively. It is known that increasing the number of streams in an XOR-APUF exponentially increases modeling complexity, which justifies the large increases in numbers of CRPs. Each k-XOR-APUF was attacked with varying amounts of CRPs and noise levels by Multilayer Perceptron. Noise levels included 0%, 5%, 10% and 20%, with a majority vote mechanism implemented on k-XOR-APUFs with a noise level of 20%.

XOR-APUF streams	CRPs used	Noise Levels
4-XOR	1,000,000	0% noise 5% noise 10% noise 20% noise 20% noise with Majority Voting
5-XOR	1,000,000	
6-XOR	2,000,000	
7-XOR	2,000,000	
8-XOR	5,000,000	
9-XOR	5,000,000	

Table 3: Challenge-Response-Pair (CRP) counts used for k-XOR-APUF modeling across different noise levels.

Attacking Interpose PUF

The iPUF configurations attacked were $(1, k)$-iPUF and (k, k)-iPUF, where k is the number of streams in each XOR-APUF layer and ranged from 4 to 7 inclusively. Each configuration of iPUF was of 64-bitlength and was attacked under noise-free conditions to solely focus on the robustness of the ML models without the added complexity introduced by noise. Attacks on iPUF seen the highest number of CRPs needed. For example, attacking the $(7, 7)$-iPUF required 40 million CRPs.

Due to time constraints and resource limitations, only the $(1, k)$-iPUF configurations were fully modelled, that is, the upper and lower layers were fully modeled and incorporated into one model, depicting the fully attacked Interpose PUF. This was made possible by the fact that the upper layer in these configurations is a 1-XOR-APUF, acting as a regular Arbiter PUF which can be easily attacked by LR with a small number of CRPs. The training data for the upper layer was obtained through the heuristic in **Algorithm 1**. Attacking the (k, k)-iPUF configuration presented a greater challenge due to

the increase in non-linearity in the upper layer, therefore requiring more CRPs to model the upper layer.

iPUF Configuration	CRPs used
(1, 3)-iPUF	400,000
(1, 4)-iPUF	400,000
(1, 5)-iPUF	800,000
(1, 6)-iPUF	5,000,000
(1, 7)-iPUF	10,000,000
(3, 3)-iPUF	400,000
(4, 4)-iPUF	1,000,000
(5, 5)-iPUF	10,000,000
(6, 6)-iPUF	20,000,000
(7, 7)-iPUF	40,000,000

Table 2: Challenge-Response-Pair (CRP) counts used for different iPUF configurations, (1, k) and (k, k).

5.2 Attacking Model

Modeling attacks were conducted on the three PUFs mentioned in this paper; Arbiter PUF, XOR-Arbiter PUF and Interpose PUF. The experiments were carried out on a MacBook Air equipped with an M1 chip with a 3.2 GHz max. clock rate and 16 GB of memory. To evaluate the attacking model, the unseen testing CRP's prediction accuracy was measured. Additionally, to ensure that the models were trained on balanced data and not biased toward a particular response class (i.e., 0's or 1's), the entropy of each dataset was examined prior to training. This was done by calculating the uniformity of the response distribution, using Hamming Weight as an estimate of balance between the response classes, defined as:

$$U = \frac{1}{C}\sum_{i=1}^{C} r_i \times 100, \tag{11}$$

where C is the number of CRPs and r is the response bit. A uniformity close to 50% indicates a well-balanced dataset, which is ideal for binary classification. The ideal rate of the uniformity is 50%, where it indicates a 50% of 0's and 1's in a dataset. The following three tables outline the average uniformity for Arbiter PUFs, XOR-Arbiter PUFs and Interpose PUFs for different configurations.

Arbiter PUF bitlength	Average Uniformity
64	50.1%
128	50.1%

XOR-APUF size	Average Uniformity
4-XOR	50.0%
5-XOR	50.0%
6-XOR	50.2%
7-XOR	50.0%
8-XOR	50.0%
9-XOR	50.0%

iPUF configuration	Average Uniformity
(1, 3)-iPUF	49.3%
(1, 4)-iPUF	50.4%
(1, 5)-iPUF	50.0%
(1, 6)-iPUF	50.0%
(1, 7)-iPUF	50.0%
(3, 3)-iPUF	50.4%
(4, 4)-iPUF	50.4%
(5, 5)-iPUF	50.0%
(6, 6)-iPUF	50.0%
(7, 7)-iPUF	50.0%

6.0 Experimental Results

6.1 Arbiter PUF Modeling Accuracy Results

Although the modeling of an APUF is not a difficult machine learning problem, the APUF was still attacked with differing parameters to demonstrate the effect of noise and bit length against the success of modeling an APUF. **Table 6** shows the accuracy rate achieved by LR on the Arbiter PUF with 64 bitlengths and 128 bitlengths. Non-noisy APUFs demonstrated severe modeling vulnerability with near 100% accuracy achieved for both the 64-bit and 128-bit configurations, although this was expected due to the linearity of the APUFs. The experiments ran with up to 100,000 CRPs, but a very high prediction accuracy rate was still achieved around the 10,000 CRPs mark, as shown in **Figure 7**.

Noise Level	64-bit accuracy	128-bit accuracy
0%	99.9%	99.8%
5%	94.5%	94.0%
5%†	99.6%	99.6%
10%	88.8%	89.1%
10%†	98.9%	98.5%
20%	78.7%	78.6%
20%†	93.6%	94.7%

Table 6: Predication accuracy rates achieved for 64-bit and 128-bit configurations of Arbiter PUFs. Attacks were completed using Logistic Regression with 100,000 CRPs for each noise level. Majority voting was implemented for noisy APUFs and is represented by †.

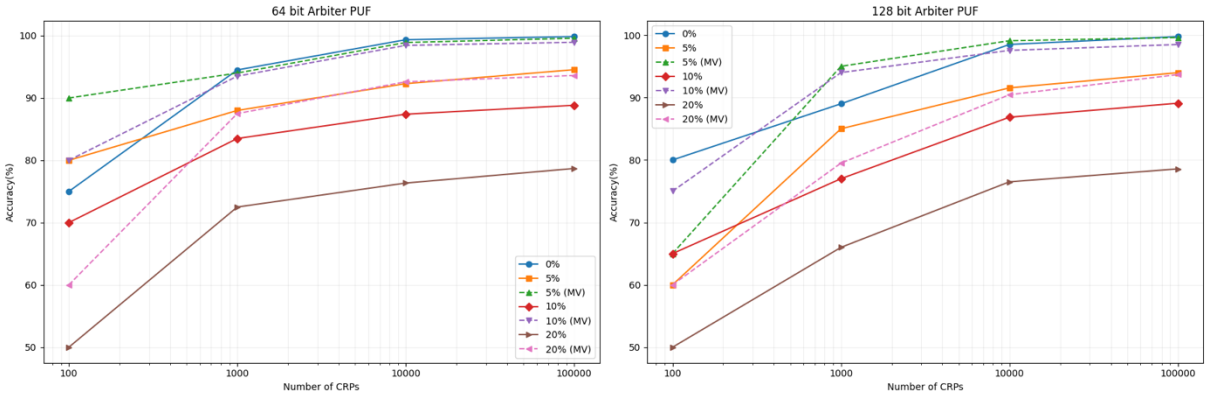

Figure 7: A visual side-by-side representation of predication accuracy for LR attacks on APUFs with differing noise levels for both the 64-bits and 128-bits configurations. Clearly seen on both charts, the number of CRPs highly influences the success of the modeling attack.

The LR model did not pick up on any details to see past the noise level, as the highest prediction for the 5%, 10%, and 20% noise datasets achieved were 94.5%, 88.8%, and 78.7%, respectively, on the 64-bit configuration. However, in the real world this would not cause an issue; if the reliability of the PUF was 80%, then as long as the model matches the reliability there would no cause for concern about counterfeiting, and perhaps if the model achieved more than the said 80% reliability then this might be an indication of tampering.

The majority voting technique proved to be effective, with the 5% noise and 10% noise APUFs achieving a 99.6% and 98.9% prediction accuracy, respectively. The APUF with 20% noise achieved 93.6% accuracy rate on the majority vote inflicted CRP dataset. Comparing these accuracies against the estimated final noise level in **Table 1**, the actual 5% and 10% noise levels achieved a *higher* prediction rate, with the 20% noise level slightly falling short. However, this difference could be attributed to insufficient CRPs for the estimated final noise level to achieve a balanced prediction.

6.2 XOR-Arbiter PUF Modeling Accuracy Results

Although XOR-APUFs were designed to break the non-linearity of Arbiter PUFs, therefore making it more difficult to attack, the XOR-APUFs were all still able to be attacked with relatively smaller numbers of CRPs compared to other studies [**5**]. Table 7 highlights the predication accuracy results for attacking k-XOR-APUFs, where k ranged from 4 to 9 inclusively:

XOR-APUF streams	Number of CRPs	0% noise	5% noise	10% noise	20% noise	20% (MV)	Average attack time
4-XOR	1,000,000	99.2%	93.6%	89.2%	79.2%	93.5%	0:24 minutes
5-XOR	1,000,000	99.4%	94.3%	89.1%	79.3%	93.5%	0:28 minutes
6-XOR	2,000,000	99.4%	94.4%	89.4%	79.4%	93.6%	1:03 minutes
7-XOR	2,000,000	99.2%	94.2%	89.1%	79.0%	93.3%	1:48 minutes
8-XOR	5,000,000	99.1%	94.0%	89.1%	50.0%	93.4%	
9-XOR	5,000,000	99.0%	93.9%	88.8%	49.9%	93.1%	

Table 7: Prediction accuracy for 64-bit k-XOR-APUF across different noise levels, with a majority voting applied for 20% noise.

Noise free XOR-APUFs (0% noise) consistently achieved prediction accuracy rates of over 99% across all stream counts. The increase of streams for the XOR-APUF exponentially increases complexity, however with enough CRPs the PUF can still be modeled to a high accuracy, as seen with the 8-XOR and 9-XOR configurations.

However, as the noise level increased, the prediction accuracies decreased. It could have been proposed that the MLP could have seen past the noise level and achieve a prediction rate higher than the reliabilities of the PUF, however this is not the case. Notably, at 20% noise without majority voting, the model's performance on 8-XOR and 9-XOR degraded substantially, achieving only around 50% accuracy, which is the same as randomly guessing. This emphasises the impact noise can have on hindering machine learning attacks if left uncorrected. Though, this could have been corrected with a higher number of CRPs. **Figure 8** shows a visual representation of the prediction rates achieved for each k-XOR-APUF and each noise level, against training epoch.

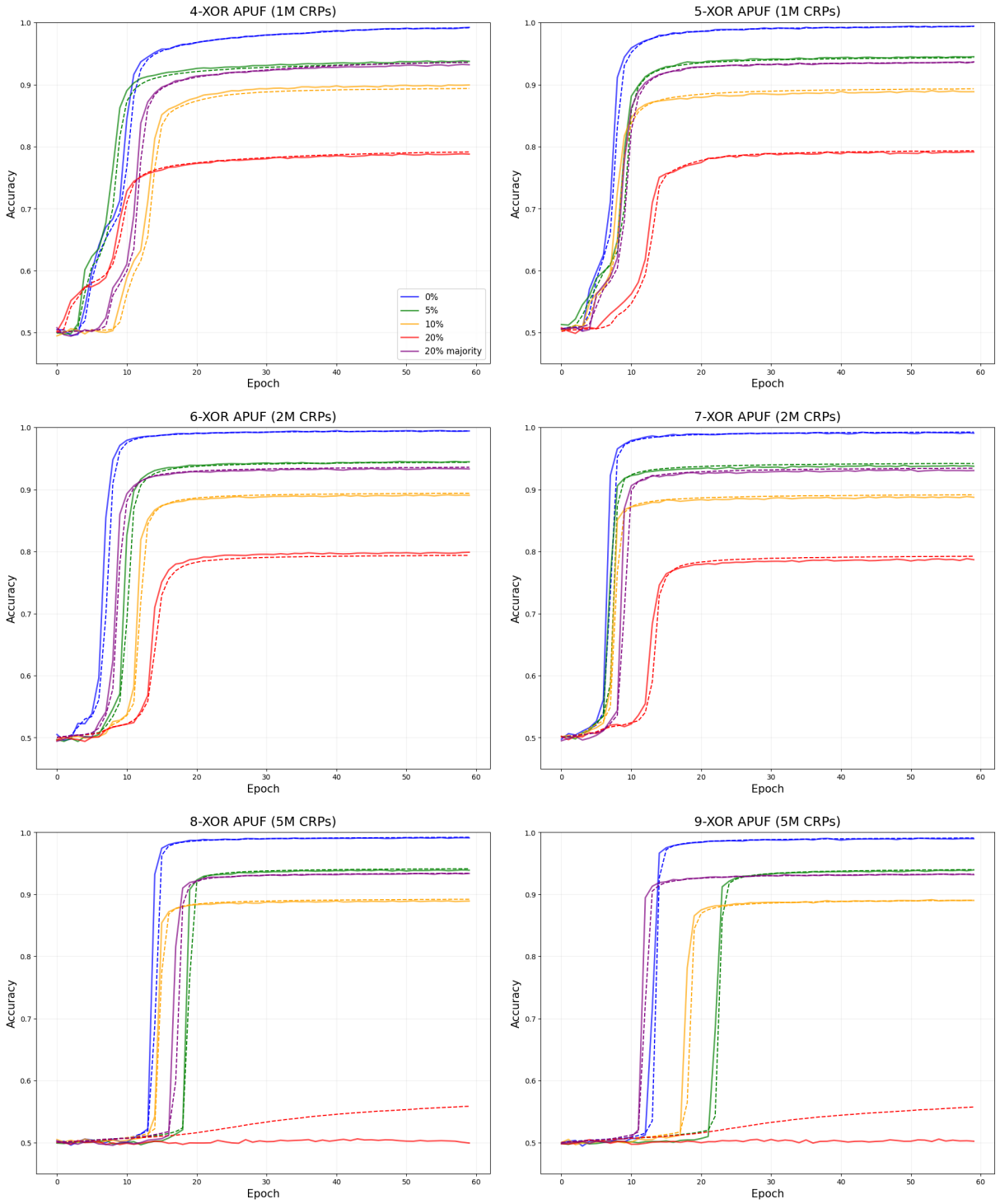

Figure 8: Prediction accuracy against training epochs for 4-XOR to 9-XOR APUFs under different noise levels (keyed in top left graph), with validation accuracy as a solid line (–) and testing accuracy as a dashed line (--). Each graph shows the model's ability to converge for different noise levels, with higher noise levels causing slower convergence and lower final prediction accuracies, and no convergence at all for 8-XOR and 9-XOR APUFs with 20% noise.

The incorporation of majority voting (MV) at 20% noise substantially improved results across all configurations. Even for the 8-XOR and 9-XOR configurations, majority voting allowed prediction accuracies to climb back to above 93%. This highlights that although noise increases the difficulty of modeling, mitigation strategies like majority voting can partially reverse this effect.

6.3 Interpose PUF Modeling Accuracy Results

Although Interpose PUFs were specifically designed to resist modeling attacks through their non-linear and layered structure, they are demonstrated to remain vulnerable to machine learning attacks given enough CRPs. **Table 8** shows the prediction accuracy achieved when attacking (1, k)-IPUF configurations with k ranging from 3 to 7:

iPUF Configuration	Number of CRPs	Lower layer accuracy	Upper layer accuracy	Full accuracy
(1, 3)-iPUF	400,000	98.4%	57.0%	98.6%
(1, 4)-iPUF	400,000	98.0%	51.0%	98.0%
(1, 5)-iPUF	800,000	97.8%	58.4%	97.9%
(1, 6)-iPUF	5,000,000	74.7%	97.9%	98.7%
(1, 7)-iPUF	10,000,000	74.1%	96.3%	97.6%

Table 8: Predication modeling accuracy for different iPUF configurations.

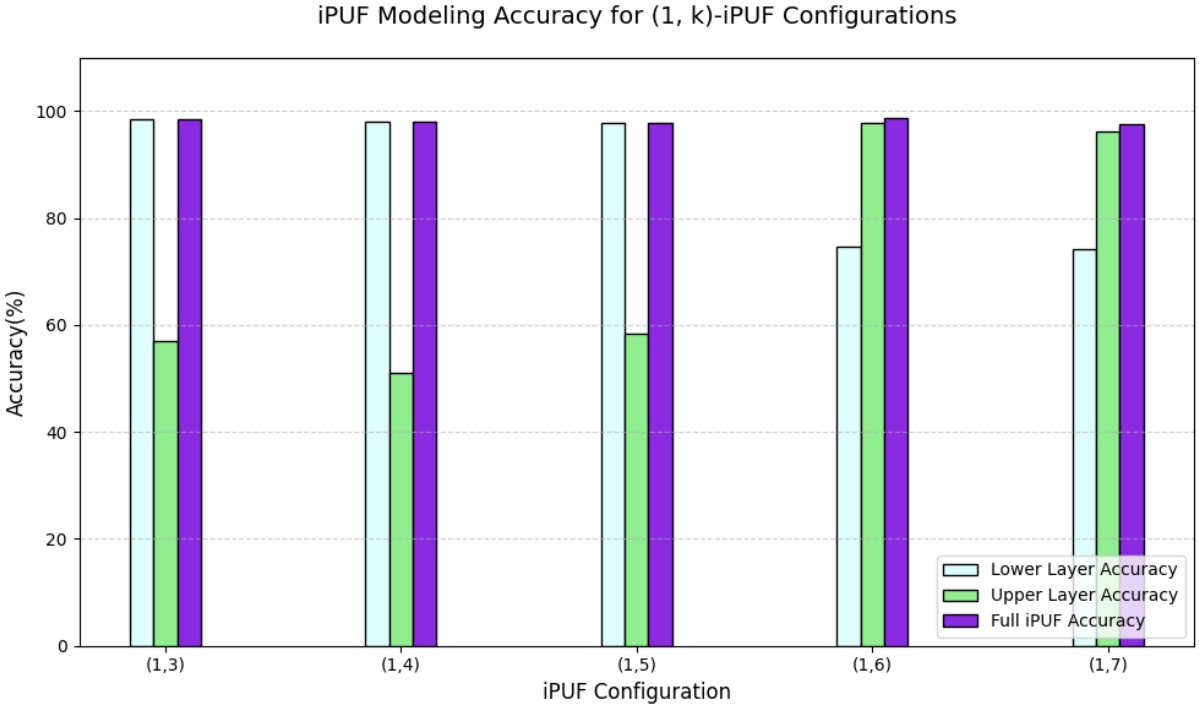

Figure 9: Bar charts showing the prediction rates for modeling the iPUF for different configurations, where the upper layer XOR-APUF only has one stream. Each configuration has a lower layer accuracy, upper layer accuracy, and a total accuracy for modeling.

As shown in **Table 8** and **Figure 9**, each (1, k)-iPUF has been modeled to a very high accuracy, with the lowest prediction accuracy obtained being 97.6% with the (1, 7)-iPUF. The full iPUF accuracy consistently stays high; however, the lower layer and upper layer accuracies fluctuate. For the (1, 3), (1, 4), and (1, 5) iPUF's, the upper layer never reached a predictive rate of over 60%; however, the lower layer accuracy reached an accuracy of >97%. For the (1, 6) and (1, 7) configurations, the lower model prediction accuracy drops; however, the upper layer accuracy increases.

For the more complex (k, k)-iPUF configurations, a high accuracy was obtained by modeling the lower layer only, but only for smaller numbers of k, as shown in **Table 9**:

iPUF Configuration	Number of CRPs	Lower layer accuracy
(3, 3)-iPUF	400,000	97.9%
(4, 4)-iPUF	1,000,000	91.2%
(5, 5)-iPUF	10,000,000	75.4%
(6, 6)-iPUF	20,000,000	75.2%
(7, 7)-iPUF	40,000,000	74.6%

Table 9: Prediction accuracy for modeling more complex (k, k)-iPUF configurations by only attacking the lower layer.

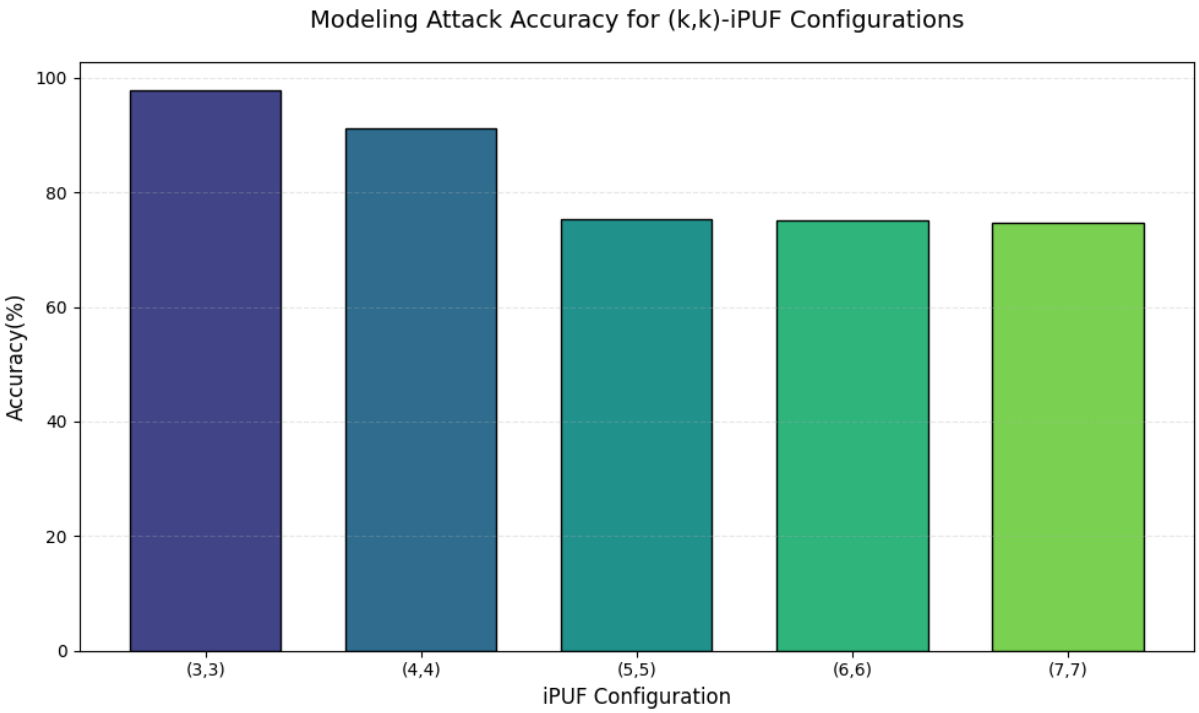

Figure 10: Bar chart showing the prediction rates for modeling the iPUF for different configurations, where the upper layer and lower layer XOR-APUFs contain the same number of streams. Attacks were conducted on the lower layer only, by inserting random bits into the challenges of the CRP dataset.

As seen in **Table 9** and **Figure 10**, the accuracy rates for (3, 3)-iPUF and (4, 4)-iPUF variants highly surpassed the theoretical accuracy rate of 75% via insertion of random bits. This means that the MLP model was able to model the behaviour of the upper layer's bit interposed into the lower layer's challenge. However, for (5, 5)-iPUF and above, this is not the case. Whether this is because of a lack of CRPs or the model not being powerful enough, the model still achieved an accuracy of 75%. As mentioned in Section 5, the (k, k)-iPUF configuration was not able to be attacked by divide-and-conquer due to time constraints, however with the addition of this strategy, the accuracy rates for the higher complexity iPUF's could have surpassed the 75% accuracy.

This only shows the increase in effort to model the iPUF, as to model it requires significantly more CRPs and a longer modeling time, taking into consideration the time it would take to model the upper

layer. Though the iPUF is not entirely resilient to modeling attacks, it is significantly better than its counterpart, the XOR-APUF.

7.0 Conclusion

The Arbiter PUF does propose a promising solution to be a lightweight device-specific identifier, but the linearity of the PUF and vulnerability to ML learning attacks with small amounts of CRPs raises concerns about security. In just matter of seconds, the APUF can be attacked to an accuracy of >99% with ≈10,000 CRPs using Logistic Regression. For the APUF to be adopted widescale, these critical matters of concern need to be addressed.

The XOR-Arbiter PUF, though more resilient in terms of effort required to attack, is no better than the APUF since it can still be modelled to a high accuracy in a relatively short amount of time. A 4-XOR-APUF can be attacked in as little as 25 seconds with 1,000,000 CRPs, which is no time at all in terms of attacking a security device. Additionally, with the increase in noise from the individual APUFs tallying up, it is difficult to find a sweet spot between complexity and reliability.

The Interpose PUF does perform better at defending against ML attacks, however an inherent 75% accuracy rate solely from the upper layer interposed bit phenomenon immediately hinders the "unclonability" of the PUF. The Interpose PUF does hold up the best however, with a dataset of 40 million CRPs just to achieve a lower layer prediction accuracy of 74.6% for the (7, 7)-iPUF configuration. If the time had been available, the (7, 7)-iPUF could have been attacked by divide-and-conquer, but this just amplifies the level of effort required to model the more complex iPUF's; since the more complex iPUF's could not be fully modeled in this paper, it highlights the PUF's resilience against ML attacks.

The level of noise in a CRP dataset has a major play in the final predictive accuracies of the ML models. It was hoped that the models could see past the noise and achieve an accuracy rate surpassing the reliability of the PUF, but this was not the case. Though the noise caused a loss in accuracy, it did not pose any threat to the machine learning models to successfully attack the PUFs, except for 8-XOR and 9-XOR APUFs. The reason these models did not converge could be for a variety of reasons, but the main thing to take away from this is with noise in a dataset, if not properly mitigated it could hinder the success of attacking the PUFs. Let's take for example the 8-XOR APUF with 20% noise; the model was not able to converge, but with a majority voting mitigation technique applied, the model converges and achieves a respectable 93.4%.

To conclude, the three types of Strong-PUFs mentioned in this paper were all able to be attacked, with the lowest predictive accuracy rate being 74.6%, a solid 24.6% higher than randomly guessing. More research needs to be conducted in the field of Physical Unclonable Functions before they will ever be adopted on a widespread scale. It is too easy to attack even the highest of complex PUFs with enough resources.

References

[1] U. Rührmair, F. Sehnke, J. S¨ olter, G. Dror, S. Devadas, J. Schmidhuber, Modeling attacks on physical unclonable functions, in: Proceedings of the 17th ACM Conference on Computer and Communications Security, 2010, October, pp. 237–249, https://doi.org/10.1145/1866307.186633.

[2] M. S. Alkatheiri and Y. Zhuang, "Towards fast and accurate machine learning attacks of feed-forward arbiter PUFs," 2017 IEEE Conference on Dependable and Secure Computing, Taipei, Taiwan, 2017, pp. 181-187, doi: 10.1109/DESEC.2017.8073845.

[3] N. Wisiol, “Splitting the Interpose PUF: A Novel Modeling Attack Strategy”, TCHES, vol. 2020, no. 3, pp. 97–120, Jun. 2020, doi: 10.13154/tches.v2020.i3.97-120.

[4] Mursi, Khalid T., Bipana Thapaliya, Yu Zhuang, Ahmad O. Aseeri, and Mohammed Saeed Alkatheiri. 2020. "A Fast Deep Learning Method for Security Vulnerability Study of XOR PUFs" *Electronics* 9, no. 10: 1715. https://doi.org/10.3390/electronics9101715

[5] Santikellur, Pranesh, Aritra Bhattacharyay and Rajat Subhra Chakraborty. “Deep Learning based Model Building Attacks on Arbiter PUF Compositions.” *IACR Cryptol. ePrint Arch.* 2019 (2019): 566. https://eprint.iacr.org/2019/566

[6] Lim, Daihyun, Jae W. Lee, Blaise Gassend, G. Edward Suh, Marten Van Dijk, and Srinivas Devadas. "Extracting secret keys from integrated circuits." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13, no. 10 (2005): 1200-1205. https://ieeexplore.ieee.org/abstract/document/1561249

[7] J. Yao et al., "Design and Evaluate Recomposited OR-AND-XOR-PUF," in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 2, pp. 662-677, 1 April-June 2022, doi: 10.1109/TETC.2022.3170320. https://ieeexplore.ieee.org/abstract/document/9766080

[8] 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”, TCHES, vol. 2019, no. 4, pp. 243–290, Aug. 2019, doi: https://doi.org/10.13154/tches.v2019.i4.243-290

[9] Mohammed El-Hajj, Ahmad Fadlallah, Maroun Chamoun, Ahmed Serhrouchni, A taxonomy of PUF Schemes with a novel Arbiter-based PUF resisting machine learning attacks, Computer Networks, Volume 194, 2021, 108133, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2021.108133.

[10] U. Rührmair and J. Sölter, "PUF modeling attacks: An introduction and overview," 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 2014, pp. 1-6, doi: https://doi.org/10.7873/DATE.2014.361

[11] G. Edward Suh and Srinivas Devadas. 2007. Physical unclonable functions for device authentication and secret key generation. In Proceedings of the 44th annual Design Automation Conference (DAC '07). Association for Computing Machinery, New York, NY, USA, 9–14. https://doi.org/10.1145/1278480.1278484

[12] Xiaolin Xu and Daniel Holcomb. 2016. A Clockless Sequential PUF with Autonomous Majority Voting. In Proceedings of the 26th edition on Great Lakes Symposium on VLSI (GLSVLSI '16). Association for Computing Machinery, New York, NY, USA, 27–32. https://doi.org/10.1145/2902961.2903029

[13] U. Rührmair et al., "PUF Modeling Attacks on Simulated and Silicon Data," in IEEE Transactions on Information Forensics and Security, vol. 8, no. 11, pp. 1876-1891, Nov. 2013, doi: 10.1109/TIFS.2013.2279798. https://eprint.iacr.org/2013/112

[14] Scikit-learn developers, "sklearn.linear_model.LogisticRegression - scikit-learn 1.4.2 documentation," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[15] C. Zhu, R. H. Byrd and J. Nocedal. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization (1997), ACM Transactions on Mathematical Software, Vol 23, Num. 4, pp. 550 - 560.

[16] J. Yao et al., "Design and Evaluate Recomposited OR-AND-XOR-PUF," in IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 2, pp. 662-677, 1 April-June 2022, doi: 10.1109/TETC.2022.3170320. https://doi.org/10.48550/arXiv.2110.00909

[17] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539

[18] Johannes Tobisch and Georg T. Becker. 2015. On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation. In Revised Selected Papers of the 11th International Workshop on Radio Frequency Identification - Volume 9440 (RFIDsec 2015). Springer-Verlag, Berlin, Heidelberg, 17–31. https://doi.org/10.1007/978-3-319-24837-0_2

[19] G. Hospodar, R. Maes and I. Verbauwhede, "Machine learning attacks on 65nm Arbiter PUFs: Accurate modeling poses strict bounds on usability," 2012 IEEE International Workshop on Information Forensics and Security (WIFS), Costa Adeje, Spain, 2012, pp. 37-42, doi: 10.1109/WIFS.2012.6412622. https://doi.org/10.1109/WIFS.2012.6412622

[20] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. https://arxiv.org/abs/1412.6980

[21] Glorot, Xavier, Antoine Bordes and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks.” *International Conference on Artificial Intelligence and Statistics* (2011). https://api.semanticscholar.org/CorpusID:2239473

[22] Python Software Foundation. Python Language Reference, version 3.10. Available at http://www.python.org