Inconsistency of Simulation and Practice in Delay-based Strong PUFs

Anita Aghaie and Amir Moradi

Ruhr University Bochum, Horst Görtz Institute for IT-Security, Bochum, Germany
firstname.lastname@rub.de

Abstract. The developments in the areas of strong Physical Unclonable Functions (PUFs) predicate an ongoing struggle between designers and attackers. Such a combat motivated the atmosphere of open research, hence enhancing PUF designs in the presence of Machine Learning (ML) attacks. As an example of this controversy, at CHES 2019, a novel delay-based PUF (iPUF) has been introduced and claimed to be resistant against various ML and reliability attacks. At CHES 2020, a new divide-and-conquer modeling attack (splitting iPUF) has been presented showing the vulnerability of even large iPUF variants.

Such attacks and analyses are naturally examined purely in the simulation domain, where some metrics like uniformity are assumed to be ideal. This assumption is motivated by a common belief that implementation defects (such as bias) may ease the attacks. In this paper, we highlight the critical role of uniformity in the success of ML attacks, and for the first time present a case where the bias originating from implementation defects hardens certain learning problems in complex PUF architectures. We present the result of our investigations conducted on a cluster of 100 Xilinx Artix 7 FPGAs, showing the incapability of the splitting iPUF attack to model even small iPUF instances when facing a slight non-uniformity. In fact, our findings imply that non-ideal conditions due to implementation defects should also be considered when developing an attack vector on complex PUF architectures like iPUF. On the other hand, we observe a relatively low uniqueness even when following the suggestions made by the iPUF's original authors with respect to the FPGA implementations, which indeed questions the promised physical unclonability.

Keywords: Physical Unclonable Function, Interpose PUF, Splitting iPUF Attack, LR Attack, ANN, Uniformity, Bias, Uniqueness, FPGA, Hardware Implementation

1 Introduction

In order to limit the secure memory usage and avoid the attacks on non-volatile memories (NVMs), physical hardware primitives, such as Physical Unclonable Functions (PUFs), have been introduced to the world of the embedded devices two decades ago [GCvDD02, LLG$^+$04, SD07]. Such functions supposedly provide a secure hardware platform (e.g., a unique secure chip fingerprint) with lightweight features such as low energy consumption and small area footprint. For instance, in silicon-based PUFs, the differences in transistors' properties, especially threshold voltage, originating from process variations (e.g., random dopant fluctuations), cause varying timing (delay) characteristics in identical circuits. Although such device-dependent properties are also the core of other PUF constructions like optical PUFs [PRTG02] or coating PUFs [Pos98, SMKT06], the silicon delay-based PUFs [GCvDD02] absorbed the lion's share of both academia and industry's attention as they do not require any analog signal or external measurement facilities.

PUFs, mostly known as chip fingerprints due to their unique inherent physical features, have various cryptographic applications in the Internet of Everything (IoE) [HYKD14, RH14, LLG+04]. For instance, their applications are shown in block-chain of devices and data security [MYKP20], in memory usage and cloud services [SG18, DBNT19], for generating a random secret key for cryptographic ciphers [SD07], or in PUF-based authentication protocols [ZBX+20, MRK+12, YMVD14, YHD+16]. The *challenge-response* mechanism is the main approach of these one-way physical functions with a unique behavior for each PUF. An m-bit unpredictable, stable, and unique response is answered to a random n-bit challenge to create a set of Challenge-Response Pairs (CRPs). Supposing a fully reliable (noise-free) case, a PUF can be seen as a Boolean function, which represents this challenge-response behavior [GTS16, GFS19]. Since such a function is not fully known, usually a set of CRPs, seen as one-time authentication tokens, is collected and stored in the server during the enrollment phase. It is worthy to recall that in some authentication protocols, instead of long CRP lists, the predictive machine-learning model of the underlying PUF (extracted by the designer) should be stored in the server [Del17]. This naturally helps compensating the area and timing overheads in various applications.

With respect to the number of CRPs and PUF architectures, these physical functions are categorized in two groups: *weak* PUFs like SRAM PUFs [MvdL14] and *strong* PUFs such as Arbiter PUFs (APUFs) [GCvDD02] and interpose PUFs (iPUFs) [NSJ+19]. Weak PUFs usually generate a limited number of secure unique responses. They mainly need to make use of an error-correction approach such as helper data and fuzzy extractors [HKM+12, DGV+16, DGSV15, DV14] to produce a highly-secure and reliable key for cryptographic applications. In contrast, strong PUFs have a large and exponential-size CRP sphere that gives them more flexibility in terms of usage, and at the same time more vulnerability to the modeling attacks. Regardless of their type, each PUF is evaluated by several metrics related to their physical characteristics such as reliability, uniformity, and uniqueness. The promise of an ideal PUF, that generates reliable tamper-evident responses to the same repeated challenges under different circumstances, is not mostly fulfilled due to several serious threats like Machine Learning (ML) attacks. In more detail, the known threats to the security of PUFs can be classified into four categories: modeling attacks [RSS+10, Del19, HMOR94, SBC19], side-channel-analysis-based modeling attacks [DV13, Bec15, RXS+14, AM20], cryptanalysis [NSMC14], and physical cloning without using any modeling learners [TDF+17]. According to the recent publications, ML attacks (the first category) are identified as the most common and effective threats, which can be performed with a classifier algorithm like Logistic Regression (LR) [RSS+10], Support Vector Machines (SVMs) [LLG+04, NSJ+19], Decision Trees (DTs), or using black-boxes learning algorithms like Artificial Neural Networks (ANNs) [SBC19, Del19]. The second category is also among the PUF designers' concerns, since the extracted physical information like reliability and power consumption associated to the CRP sets can help the attacker to simulate the PUF model through different learning algorithms such as LR and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [TAB20, Bec15, BK14]. The first and third attack categories use only CRPs as the source for building a model, while the second one uses per-challenge physical information to find models for individual components of the PUF in a divide-and-conquer manner. All kinds of attacks are intimately associated with their state-of-the-art choice of the algorithm and the category being used interchangeably.

PUF designers find the hardware implementation of these physical functions adequately challenging [MKD10, MKKD14]. The most difficult challenge is to achieve an implementation with physical characteristics not far from ideal. Along the same line, delay-based PUFs are prone to environmental noise like temperature and supply voltage variations. This motivated the researchers to deal with various hardware implementations of delay-based PUFs in the form of Ring Oscillator PUFs (RO PUF) and Arbiter PUFs (APUF) on Field-

Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuit (ASIC) platforms [MS11, RSS$^+$13, MvdL14, SCM15]. Owing to the fact that the functionality of APUFs and XOR-APUFs is based on a tiny delay difference between lines that should only be affected by process variations, the implementation of these PUF primitives encounter difficulties to achieve a symmetric structure/routing (particularly in FPGAs). Further, APUF and RO PUFs play a primary roll in different strong PUF constructions, and are applied as a basic PUF primitive in larger designs. Therefore, all such implementation difficulties hold valid (and even become more challenging) when dealing with composite constructions. As a result, most of the aforementioned ML attacks are usually applied and investigated purely in the simulation domain since implementation defects such as bias tend to make the learning problem easier. To exclude the possibility that a hidden defect may have enabled an otherwise non-functional attack, the analyses are usually started by (and most of the time are limited to) software-simulated models. Another reason behind such a pure software-based analysis, is to ensure the reproducibility and comparability of the results, i.e., a common baseline for the hardness of the learning problem is provided this way.

The most recent composite construction, i.e., (x, y) iPUF, combines two XOR-APUF layers with x and y APUF instances, respectively [NSJ$^+$19]. Different iPUF instances have been evaluated by various pure ML attacks like LR, CMA-ES, ANN, and also by the reliability attacks (a form of SCA-based modeling attacks). Apart from the authors' claims on the resistance of their construction against several attacks, state-of-the-art classifier modeling attacks and ANN have been successfully applied on various iPUF instances in the simulation domain. For example, in [SBC19] the authors reported the modeling of up to (4,4) iPUF variants using Multi-Layer Perceptron (MLP) with three layers and Adam optimization. Moreover, a novel modeling attack, known as splitting iPUF [WMP$^+$20], is reported to be able to model the iPUF by means of splitting the learning process to two phases and making use of LR as the learner. It is also demonstrated in [TAB20] that the claimed-impossible reliability attack [Bec15] can achieve a high modeling accuracy using a new optimization strategy.

1.1 Related works

Various implementations of APUFs have been examined on different platforms in several works [MKD10, MKKD14, KKR$^+$12, SNCM16, MRV$^+$12, GTS$^+$19, SCM15, SMCN18]. The concern of accurate APUF implementation, which is only affected by physical/process variation, emerged in [MKD10, MKKD14]. The authors suggested to use programmable delay lines (tuner blocks) for identical APUF implementation on FPGAs. In [KKR$^+$12], five PUF instances, including APUF, have been explored on ASICs to evaluate their entropy and reliability with temperature variation. In addition, another study analyzed the reliability of APUFs and their uniqueness in ASICs accompanying the effect of aging [MRV$^+$12].

However, all these implementations focus on reliability and uniqueness of APUFs. They have mostly compared APUF with other basic primitives. To the best of our knowledge, only a few works including [SNCM16, WP20] have studied the inherent architectural bias of delay-based PUFs in particular APUF-based constructions. In [SNCM16], the authors evaluated 1000 PUF instances to highlight the architectural bias and the implementation bias in various APUF architectures, but again it has been conducted purely in the simulation domain (Matlab simulation). As a summary, delay-based PUFs are accounted as architectures with lower bias compared to other PUF primitives, e.g., dual APUF (DAPUF) [MYIS15] and programmable line PUF (PAPUF) [MKD10, MKKD14, SNCM16].

Further, the effect of *bias* (also called *balance*) in the input data sets (i.e., here is CRPs sets) has been studied in the ML community [NJGS09, CM91, Bis07]. The selection of an appropriate learner algorithm is considered as an important task in ML attacks to i) yield an optimized success rate, and ii) observe that the learner behaves properly when encountering biased data sets. For instance, LR algorithms can occasionally suffer from the biased learning which might be mitigated by choosing a proper regularization or small sample sizes as suggested in [NJGS09, KZ01].

1.2 Our Contributions

The primary motivation of this work is that besides the PUF architectures, the metrics like uniformity, uniqueness, and reliability can affect their robustness against modeling attacks. Our main focus is on the uniformity metric that affects the learning process through the most-commonly-used ML classifier, i.e., LR, which is also used in the splitting iPUF attack [WMP+20]. As stated, ML attacks are usually evaluated only by software simulations as implementation defects such as bias may ease the learning process. In this paper, we for the first time show that implementation defects can inversely make certain learning problems harder in the complex PUF models. We demonstrate that even a slightly non-uniform CRP set causes a loss in modeling accuracy in the training process, although several optimizations can moderately compensate for it. These analyses are conducted on both classical LR (suggested by the iPUF's original authors) and splitting iPUF attacks.

In short, we perform our investigations on real data sets collected from more than 1000 various iPUF implementations (of the same design placed and routed differently) on 100 FPGAs of the Xilinx Artix 7 family. These data sets help us to precisely analyze the APUFs metrics on a large scale. More importantly, this allows us to observe the effect of uniformity of APUFs on other terms like uniqueness and on their robustness against ML attacks. These findings, confirmed by our experimental study, motivated us to expand our analyses by adjusting and applying other ML attacks to the iPUF. In order to tolerate the non-uniformity (originating e.g., from implementation defects), we optimize the splitting iPUF attack by considering bias in the LR algorithm, interfering the sample size, and applying another learner like ANN. We further investigate the uniqueness of our case study on the aforementioned FPGA cluster. We also try to find out whether it is straightforward and affordable for the attacker to make a single model from a PUF's actual FPGA implementation thereby precisely predicting the response of other PUF instances.

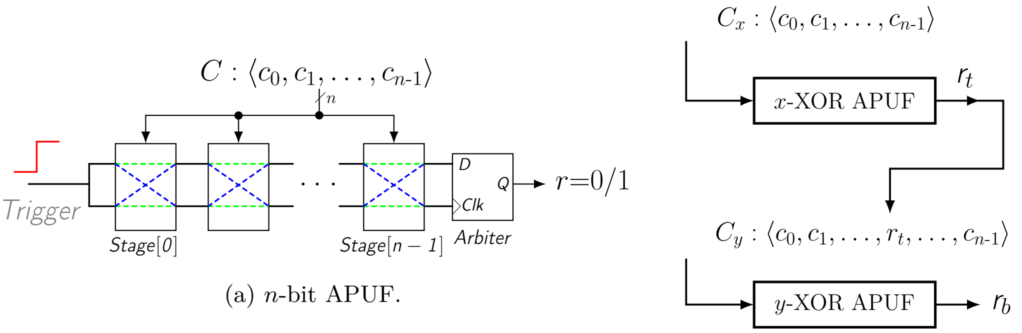
2 Preliminaries

This section gives a brief background on APUF-based architectures and applied ML attacks, including the new splitting iPUF attack. We shortly describe the PUF metrics and parameters, which we use in our analyses by investigating their effects on modeling attacks.

Notations. We denote binary or real random variables $x \in \mathbb{F}_2 / \in \mathbb{R}$ with *italic*, vectors $X \in \mathbb{F}_2^{n>1} / \in \mathbb{R}^{n>1}$ with *CAPITAL italic*, elements in a vector with superscripts x_i, and functions $\mathsf{f}(.)$ with *italic sans serif* font.

2.1 APUF, XOR-APUF, and iPUF

One of the most prevalent strong PUF primitives is APUF, which is applied in many delay-based PUF architectures [LLG+04, GCvDD02, SD07, NSJ+19]. This lightweight PUF primitive is triggered by an electrical signal to traverse through two challenge-dependent paths as a race. The signals should pass n stages, each of which is controlled by a challenge bit $c_{i<n}$ as shown in Figure 1a. At the end, an arbiter (a flip-flop) decides the final response of the APUF depending on which delay line is more extensive than the other one. The commonly-used linear delay model, shown in Equation (1), perfectly models the function


$C : \langle c_0, c_1, \ldots, c_{n\text{-}1} \rangle$
n
Trigger
Stage[0]
Stage[$n-1$]
D
Q
Clk
Arbiter
r=0/1
$C_x : \langle c_0, c_1, \ldots, c_{n\text{-}1} \rangle$
x-XOR APUF
r_t
$C_y : \langle c_0, c_1, \ldots, r_t, \ldots, c_{n\text{-}1} \rangle$
y-XOR APUF
r_b

(a) n-bit APUF.

(b) (x, y) iPUF.

Figure 1: Detailed architectures of APUF and iPUF.

of an APUF.

$$\Delta = w_0\Phi_0 + \ldots + w_i\Phi_i + \ldots + w_n\Phi_n = W^T \cdot \Phi, \text{ where} \tag{1}$$

$$\Phi_{i\in\{0,\ldots,n-1\}} = \prod_{j=i}^{n-1}(1 - 2 \cdot c_j), \quad \Phi_n = 1 \tag{2}$$

The vector of weights $W : \langle w_0, \ldots, w_n \rangle$ represents the physical characteristics of the underlying APUF stages and W^T stands for W transposed. These physical characteristics are the main target of the learner functions in ML attacks. The parity vector $\Phi : \langle \Phi_0, \ldots, \Phi_n \rangle$ is applied as the feature vector in Equation (1), which is derived from the given challenge $C : \langle c_0, \ldots, c_{n-1} \rangle$ as shown in Equation (2). In such a model, the final response bit $r = 0/1$ is generated based on the delay difference Δ given to the unit sign function, i.e., $r = \Theta(\Delta)$.

In order to increase the complexity of this mode and boost their ML resistance, several delay-based PUFs have been made with APUF as a fundamental element, e.g., XOR-APUF [SD07], Feed Forward PUF [LLG+04], Multiplexer-Based APUF [SMCN18], and Bistable Ring PUF [CCL+11] with its twisted version [SH14]. Such strong PUFs promise to provide a tamper-evident feature. It means that any external temptation should lead to a small change in the PUF response and take down the original PUF behavior [NS14]. For example, changes in supply voltage can create noise for the PUF and affect its performance to be biased. In other words, this feature of PUFs is considered as noise sensitivity, which provides the strict avalanche criterion [GTS18, DGSV14].

An XOR-APUF combines at least two n-bit APUF components with an XOR gate to generate the final response [SD07]. In a similar approach, the aforesaid linear model is extended to cover XOR-APUFs. As shown in Equation (3), the parallel attributes for a k-XOR-APUF are multiplied involving k weight vectors of individual APUF [RSS+10].

$$r = \Theta\Big(\prod_{i=1}^{k}\Delta_i\Big) = \Theta\Big(\prod_{i=1}^{k} W_i{}^T \cdot \Phi_i\Big) \tag{3}$$

To make a proper trade-off between ML robustness and reliability, interpose PUF (iPUF) employs APUFs as a primitive component to make a multi-layer XOR-APUF [NSJ+19]. The most common analyzed iPUF consists of two XOR-APUF layers, (x, y) iPUF, as shown in Figure 1b. The security of this construction relies on the position of the interpose bit (ideally at $\frac{n}{2}$), the length of challenge bits n, and the number of applied APUFs in each layer x and y.

More precisely, the one-bit response of the top layer (n-bit x-XOR-APUFs), so-called r_t, plays the role of the interpose bit for the challenge set of the bottom layer. Consequently,

$n+1$ challenge bits $\langle c_0, \ldots, c_{\frac{n}{2}-1}, r_t, c_{\frac{n}{2}}, \ldots, c_{n-1}\rangle$ are given to the bottom layer ($n+1$-bit y-XOR-APUF) to generate the single bit final response r_b. Naturally, the linear model in Equation (3) can also be employed to model an iPUF, when each layer is considered as an individual XOR-APUF.

2.2 Modeling Attacks

Here, we briefly review the basics of LR and ANN.

2.2.1 Logistic Regression

Among supervised learning algorithms, LR is commonly applied in the PUF domain as it solves the classification problems and is available in an adapted version suitable for PUF applications. The goal of LR, like other modeling algorithms, is to find the best fit and optimized models that are able to describe the relationship between the output (dependent variable: here known as PUF responses) and a set of independent variables (here known as PUF challenges) [JLS13]. This algorithm assigns input data to a discrete set of classes, which are the binary sets (0/1) in the case of a PUF with 1-bit response [Bis07, RSS+10]. LR learns the APUF's weights W so that each challenge C has a probability $p(C, r|W)$ to produce 1-bit response r as presented in Equation (4). To classify the given challenges to possible responses 0 or 1, this algorithm applies an activation function (as outlined in Equation (5)) for the weight vector W, representing the inherent PUF's characteristics. Note that here $\mathsf{f}(W, C)$ refers to the predicted PUF's output, i.e., the sign function over the APUF's linear model given in Equation (1).

$$p(C, r|W) = r\sigma\big(\mathsf{f}(W, C)\big) + (1-r)\Big(1 - \sigma\big(\mathsf{f}(W, C)\big)\Big), \text{ where} \tag{4}$$

$$\sigma(x) = (1 + e^{-x})^{-1} \tag{5}$$

To find out how the predicted responses are close to the real ones, an error function is defined by taking the negative logarithm of the likelihood, as shown in Equation (6). It is also called a binary cross-entropy error function in the following form. Since the analytical solutions are not easily able to determine the optimal minimum error function value or the final ideal weight vector as $\hat{W}$, it should be optimized iteratively by taking the gradient of the error function with respect to W in Equation (7).

$$\mathsf{E}(W) = -\ln p(C, r|W) = -\Big(r \ln \sigma\big(\mathsf{f}(W, C)\big) + (1-r)\ln(1 - \sigma\big(\mathsf{f}(W, C)))\Big) \tag{6}$$

$$\nabla \mathsf{E}(\boldsymbol{W}) = \big(\sigma\big(\mathsf{f}(W, C) - r\big)\big)\Phi \tag{7}$$

It is worth to mention that maximum likelihood can show severe over-fitting for data sets that are linearly separable [Bis07]. Since the maximum likelihood solution is applied, when the hyperplane corresponding to $\sigma = 0.5$ occurs, the decision space is separated to the two classes and the magnitude of W goes to infinity. However, these problems like over-fitting can be solved by adding a regularization term to the error function, which is explained in Section 4.1.1 in more details.

2.2.2 Artificial Neural Networks

Besides the classifier algorithms like aforementioned LR algorithm, ANNs are also able to learn the PUF models as shown in [SBC19, WMP+20, NSJ+19, CDS18]. In an ANN, a connected network of artificial neurons is responsible for building the connections and making a decision based on input neurons. This network consists of an input layer, an output layer, and several hidden layers depending on the network application [HMOR94]. A cost function that can be chosen diversely calculates the difference between the outputs until the trained model achieves the minimum prediction error. An MLP which is a class of feedforward-ANN, has a fully-connected nodes with the determined weights. These ANN models use different hyperparameters for their application on the CRP data sets. For instance, MLP models apply resilient backpropagation (RMSprop) optimization as a training algorithm, or the Adadelta, Adam, and Nadam optimizations. Based on various criteria such as network applications, data sets characteristics, and running time, the proper optimization function is selected.

2.3 Splitting iPUF Attack

It has been shown that an (x, y) iPUF can be best modeled by an $x + y$-XOR-APUF (also called linearization attack achieving at most 75% accuracy [NSJ$^+$19]), and that large iPUF variants cannot be precisely modeled by black-box ANNs [SBC19]. In contrast, splitting iPUF attack applies a divide-and-conquer method to successfully break different iPUF variants [WMP$^+$20]. In a nutshell, it divides the (x, y) iPUF into two XOR-APUFs and then employs the adopted LR algorithm to these functions in an alternative repeated manner. At first, the attack starts to achieve a rough model for the bottom y-XOR-APUF with the *uniform-random* selection of the interpose bit, which should be generated by the top layer. When the LR-based learning of the bottom layer achieves an adequate but not a high accuracy of 65%, the attack initiates the next phase. Then, the learner filters the CRPs by considering only those for which a gesture for the interpose bit leads to a match between the actual response and the learned model. Afterward, the attack continues to learn only the top x-XOR-APUF to achieve high accuracy. Then, the learning process of the bottom layer is repeated but this time with the interpose bit predicted by the modeled x-XOR-APUF. In short, by means of its iterative manner, the vulnerability of $(1, k)$ and (k, k) iPUF constructions up to $(1, 9)$ and $(8, 8)$ are presented, respectively.

Further, the authors made a new technical observation that the LR-based PUF modeling algorithm [RSS$^+$10] can fully or partially recover XOR-APUFs even in the presence of feature-noise in the training set. Considering the weights correlation and the effect of the interpose bit on the half of the weights of the bottom layer are the key points of the work presented in [WMP$^+$20]. However, requiring massive CRP sets (to find efficient ones), and exponential increase of the required CRP sets for the large iPUF variants can be counted as disadvantages of this attack. The entire analyses presented in [WMP$^+$20] are based on data sets collected by simulation. As stated, one of our goals in this work is to practically examine its applicability on data sets collected from experimental setups.

2.4 PUF Metrics

PUF designs are evaluated based on specific metrics, which make them (un)suitable for different applications. In other words, specific metrics can determine particular applications for a PUF, e.g., a low-/medium-reliable PUF architecture would require a sort of error-correcting facility (e.g., fuzzy extractor) to be used for key generation. In the following, three important PUF metrics relevant to delay-based PUF designs are restated, which are considered in our practical analyses. Due to the lack of consistency and firm definitions used in public literature, some concepts with similar meaning might be called differently. Further, in some cases a certain metric is defined uniquely in different works. Nevertheless, in the rest of this article, we stay with the definitions given below, which are mainly the same as defined and used in [MGS11, NS14, MRV$^+$12, GFS19]. Note that the below-given definitions are adopted to our experiments, where we deal with a PUF with a single-bit response, which is the only instance implemented on every device (or let say every chip). All estimations are done by means of l samples (i.e., the number of CRPs) collected from k devices.

Uniformity

Considering a state-less PUF primitive with single-bit response, this parameter estimates the proportion of '1's in the PUF's response. This definition is also known as randomness [HYKS10, MGS11] and implementation bias [SNCM16]. An ideal value for uniformity is 0.5 or 50% as it reflects the unpredictability of the response of a state-less single-bit PUF primitive. Uniformity is calculated per device, although an average over uniformity of all devices can also be meaningful.

$$\text{Uniformity}_i = \frac{1}{l}\sum_{j=0}^{l-1} r_{i,j}, \tag{8}$$

where $r_{i,j}$ denotes the PUF's response to the j-th challenge when examining the i-th device with $i \in \{0,\ldots,k-1\}$ and $j \in \{0,\ldots,l-1\}$. As a side note, if the estimated uniformity is higher than 0.5, sometimes $1 - \text{Uniformity}_i$ is used, but in this work we always count the number of '1's.

Uniqueness

One of the essential PUF's evaluation metrics is the uniqueness of the response of different devices to the same challenge, which somehow reflects the physical unclonability. The intrinsic random physical characteristic of PUFs should create unique and unpredictable responses in an ideal case. Therefore, the uniqueness is defined as the average Hamming distance (HD) between responses of two distinct devices to the same random given challenges. In an ideal situation, for the same given challenges, half of the responses of two devices should be unequal, implying the ideal uniqueness to be 0.5 or 50%.

$$\text{Uniqueness} = \frac{1}{l \cdot \binom{k}{2}} \sum_{j=0}^{l-1}\sum_{i=0}^{k-2}\sum_{i'=i+1}^{k-1} \text{HD}(r_{i,j}, r_{i',j}) \tag{9}$$

Reliability

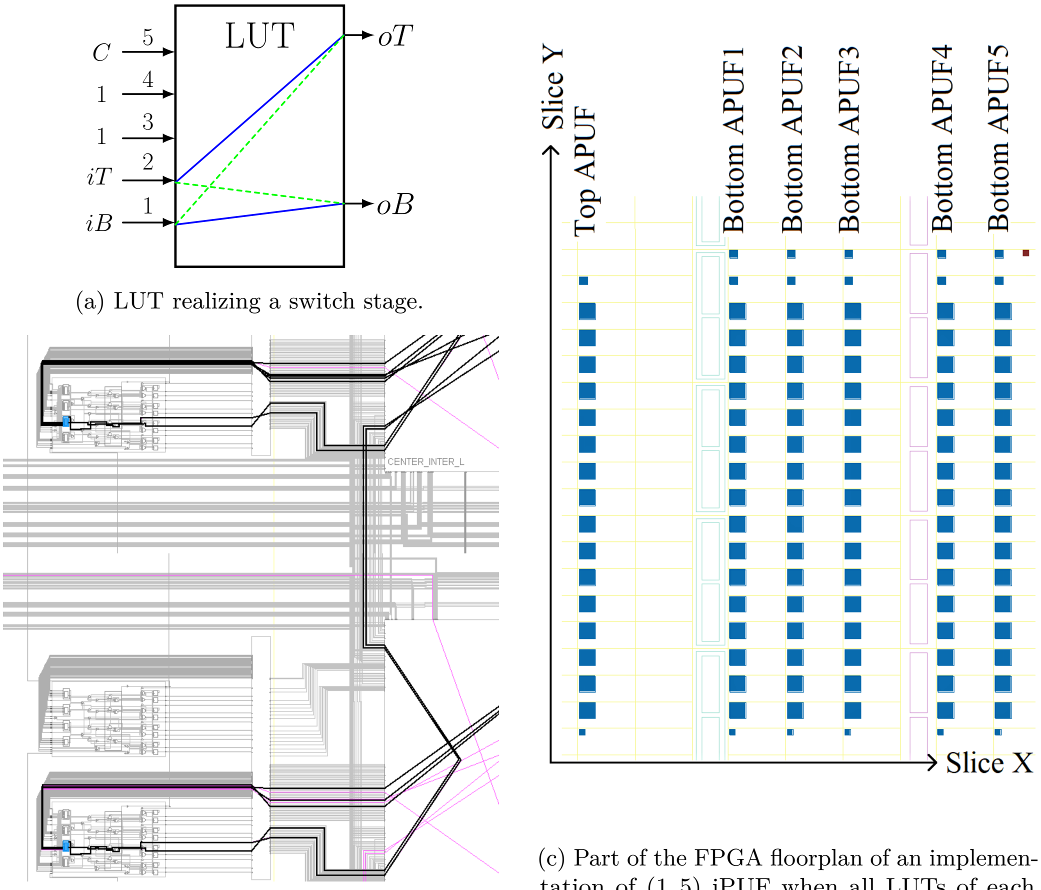
This property presents the difference between the responses of a PUF to the same repeatedly-given challenge. In other words, how reliably a PUF instance regenerates the same response to the same challenges. This metric's ideal value is 1 or 100%, which means that not only the PUF instance but also the environmental conditions should be ideal. Achieving this ideal situation is however challenging due to the environmental noise like temperature and supply voltage variations. Similar to uniqueness, reliability is also estimated for each device individually as follows.

$$\text{Reliability}_i = 1 - \frac{1}{p \cdot l}\sum_{j=0}^{l-1}\sum_{t=0}^{p-1} \text{HD}(r_{i,j}, r_{i,j}^t), \tag{10}$$

where $r_{i,j}^t$ stands for the t-th response of the underlying PUF on the i-th device to the j-th challenge, when each challenge is repeated p number of times.

3 FPGA Implementation

As our main goal is to analyze the effect of PUF metrics on the efficiency of ML attacks, particularly in experimental domain, we need to investigate our assumptions on real hardware implementations. Being reconfigurable and fast to market, FPGA platforms are usually preferred to ASICs specially for studies which require several experiments including various designs. On the other hand, FPGA implementations face difficulties when delay-based PUFs are the target application. The challenges originate from the



(a) LUT realizing a switch stage.

(b) Placement strategy using only BEL B of each slice.

(c) Part of the FPGA floorplan of an implementation of $(1, 5)$ iPUF when all LUTs of each slice are used. Each blue square shows a slice.

Figure 2: Details of the FPGA implementation.

fact that the routing, which affect the characteristics of delay lines, cannot be entirely controlled. More precisely, in contrast to ASICs, it is almost impossible to achieve a similar routing for the path delay lines of an APUF. Applying some constraints such as fixing the placement, choosing specific LookUp Table (LUT) in each slice, hindering the synthesis tool to optimize the LUT ports, keeping the hierarchy, etc. can mitigate such challenges. However, the problem cannot completely be eliminated.

In our case study iPUF, we made use of the LUT-based implementation of APUFs, that is suggested by the original authors and provided as open-source material[1]. We should emphasize that another methodology to implement switch stages of APUFs, called Programmable Delay Line (PDL), has been suggested in [MKD10]. However, due to its low uniqueness reported in [SNCM16] and in order to stay with the original implementation, we did not employ PDLs in our experiments.

According to [NSJ$^+$19], each switch stage of an APUF is implemented by a 5-to-2 LUT, as presented in Figure 2a. Therefore, for an n-bit APUF n LUTs should be instantiated. Since the placement of such LUTs (switches) affect the corresponding routing between the consecutive switches, and hence their delay, different placement strategies have been studied and suggested in [NSJ$^+$19], which are all based on using vertically-consecutive slices. The first option is to make use of only one LUT per slice. As each slice contains

[1] https://github.com/scluconn/DA_PUF_Library

four LUTs (called BEL A, B, C, and D), one strategy is to place all LUTs at the same BEL, resulting in 4 different placements for the entire APUF. Figure 2b shows part of the FPGA floorplan when only BEL B is used. Alternatively, one of the BELs can be randomly selected for each LUT in a slice, i.e., 4^n cases. We refer to this strategy as "random placement". The next option considered in [NSJ+19] is to place 2 LUTs in each slice, by fixing the pattern of the used BELs for all slices to either 'AB', 'AC', 'AD', 'BC', 'BD', or 'CD', i.e., 6 different placements. The final option is to occupy all LUTs in each slice, meaning that all BELs are used. In short, the authors stated that random placement leads to low uniformity and referred to such designs as *bad implementations*, and considered the other 11 "pattern placement" strategies as *good implementations*.

3.1 Setup

Our experimental setup consists of 100 instances of Digilent Basys3 boards, where a Xilinx Artix-7 XC7A35T FPGA is integrated. Figure 11 (in Appendix) shows a photo of the employed customized FPGA cluster. During the experiments, the FPGA cluster was being operated in a laboratory, where the temperature was kept at approximately 21°C. It is worth to mention that the machine has a controller which enables only one board at time[2], meaning that the experiments conducted on a board is not affected by the activities of the other boards. This helps keeping the environmental noise low and avoiding the boards to heat up. Hence, the surface boards' temperature is expected to be almost constant.

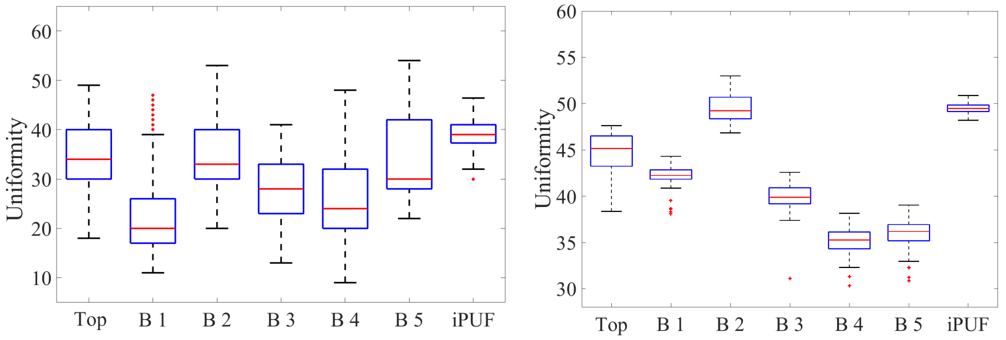
As the case study, we focused on 64-bit $(1, 5)$ iPUF as the target PUF primitive, i.e., a 64-bit APUF in the top layer, and a 65-bit 5-XOR-APUF in the bottom layer with the interpose bit in the exact middle of the challenge of the bottom layer. We followed the above-explained placement strategies and constructed 1011 implementations, i.e., FPGA bitstreams. 1000 with random placement and 11 with the suggested pattern-based strategy. In order to be able to study the behavior of each APUF instance, we intentionally routed out their output. Note that no internal wires are connected to the FPGA output drivers. The design has a state machine and communicates with a PC via UART. It means that by sending a 64-bit challenge to the FPGA, a 7-bit response is received containing the output of all APUFs concatenated with that of the iPUF. It is worth to highlight that the APUFs' output are taken from the arbiters, i.e., after being stored. In iPUF, the bottom layer is triggered when the interpose bit is ready, i.e., after the evaluation of the top layer is accomplished. Hence, connecting some internal wires to APUFs' output does not affect the functionality or characteristics of the design.

3.2 Initial Analysis

By focusing on a single device (i.e., one of those 100 FPGA boards) we examined all aforesaid 1011 designs, collected 10 k CRPs where challenges are selected uniformly at random, and extracted the uniformity of all APUF instances as well as the iPUF's final response individually for each design. Apart from all other analyses conducted on 100 FPGA boards, this initial step, whose goal is to select a few designs among 1011 ones, is done using one FPGA board. As shown later in Figure 9, the FPGA boards show a relatively low uniqueness. Therefore, repeating this initial step on multiple FPGA boards would not lead to significantly-different results.

Figure 3a represents a box plot of the extracted uniformities. As expected, the placement strategies heavily affect the uniformity of each APUF, while that of the final response stays between 33% to 47%. We also observed a relatively normal distribution for the uniformity of each APUF. Table 1 lists the uniformity of four finalists. For each placement strategy (random or pattern), we selected the first finalist (rand1/pattern1) as the one with the

[2] The power of inactive boards is cut.



(a) Uniformity of APUF instances and final response of 1011 $(1, 5)$ iPUF designs implemented on an FPGA. Estimations have been done by sets of 10 k CRPs.

(b) Uniformity of APUF instances and final response of the chosen $(1, 5)$ iPUF design implemented on a cluster of 100 FPGAs. Estimations have been done by sets of 1 million CRPs.

Figure 3: Estimated uniformities based on experimental studies.

best average uniformity over all APUF instances while keeping the uniformity of the final response in the range of 40% to 60%, and the second finalist (rand2/pattern2) as the one with the best uniformity for the final response. This range indeed follows the suggestions made in [NSJ+19]. However, as shown in Table 1, we have not found any design with the uniformity of all APUF instances being in the given range.

Table 1: Uniformity of four most favorite design candidates among random and pattern placement strategies.

Design	Uniformity						
	Top	Bottom1	Bottom2	Bottom3	Bottom4	Bottom5	iPUF
rand1	43%	40%	48%	37%	31%	29%	46%
rand2	38%	18%	33%	37%	41%	27%	47%
pattern1	21%	20%	18%	17%	70%	32%	60%
pattern2	18%	21%	9%	11%	61%	18%	53%

Finally, we have chosen the design identified as **rand1** in the first row of Table 1. Using the aforesaid FPGA cluster, we evaluated the chosen design on all 100 devices while collecting 1 million CRPs from each device. Note that, during this process, we gave the same set of challenges to all devices. In order to give an overview on the performance of our setup, collecting each 1 million CRP sets took around 18 minutes. Figure 3b depicts a box plot of uniformities extracted from these 100 sets of CRPs. We can again see an almost normal distribution for the uniformity of each APUF instance being in the range of 31% to 54%. More importantly, the uniformity of the final response (iPUF) keeps its good range of 45% to 55%. As a result, we continue most of our experimental studies, detailed in the following sections, with the chosen design, i.e., **rand1**.

4 Uniformity

The most critical requirement for delay-based APUFs is an appropriate geometry, i.e., an identical placement and routing of delay stages. As a result, the delay lines (modeled in Equation (1)) should only be influenced by the device-level deviations originating from process variations. In ideal implementations of an APUF-based composed architecture, the APUF primitives should enjoy the outstanding uniformity of 50% and uniqueness of 50%

with the highest reliability of 100%. As stated in [SNCM16, WP20], uniformity can be affected by differences in the PUF's architecture. Further, confirmed by our experimental analyses, uniformity is among the parameters sensitive to the implementation variations as well.

In this section, by focusing on our case study iPUF, we represent our practical observations regarding the effect of non-uniformity on ML attacks. Afterward, using various simulated non-uniform data sets, we show the results of the splitting iPUF attack, which is still the only state-of-the-art pure ML attack on iPUF. The section is continued with a brief theoretical explanation to point out how bias affects the LR algorithm. Then, we investigate several optimizations on this attack with the goal of maintaining the high accuracy (above 90%) in presence of non-uniformity. After that we also examine general cases such as considering various individual uniformity for each APUF and evaluating another structure like XOR-APUF. Finally, we show the result of applying the pure ANN attack as well as an ANN-adopted version of splitting iPUF attack to achieve a high modeling accuracy.

Note that, we have taken the open-source pypuf library[3], provided by the original authors of the splitting iPUF attack. Based on the same library, we developed our ANN-adopted variant fitting to different iPUF constructions. For the experimental analyses on $(1,5)$ iPUF, we used at least 1 million CRPs, while – according to [WMP+20] – a splitting iPUF attack on such an iPUF variant should require around 500 k CRPs. In order to have an insured margin in our investigations and not focusing on the scalability of the attack (that has already been done with ideal uniform data sets in [WMP+20]), we have also considered at least 1 million as the CRP-set size in our simulation-based analyses.

4.1 Splitting iPUF Attack

Based on the simulation results given in [WMP+20], the splitting iPUF attack should be able to successfully (i.e., with the success rate of 1 and the prediction accuracy above 95%) break $(1,5)$ iPUF using 500 k either noise-free or noisy CRPs. We conducted this attack on our 100 FPGA devices realizing the corresponding iPUF variant, using the design chosen in Section 3.2. It is worthy to mention that we observed a reliability of around 96% in our setup (explained in more detail in Section 5). In addition to those slightly-noisy 1 million-per-device set of CRPs collected and used to estimate the uniformity (see Section 3.2), we collected noise-free CRP sets as well. To this end, we repeatedly gave the same challenge set 11 times to each device, and extracted more-reliable CRP sets through majority voting. As a side note, collecting these 100 CRP sets took around two weeks using our setup. We then performed the original splitting iPUF attack on our noisy and noise-free CRP sets leading to the accuracy of at most 72% as shown in Figure 4. Note that we have not observed a perceptible difference between the attack results for the noisy and noise-free CRP sets. For each set of CRPs, we repeated the attack 10 times; the best results are presented in Figure 4. As a reminder, the uniformity of the final response in these CRP sets is in the range of 46% to 52%, as shown in Figure 3b, i.e., a very slight non-uniformity, while the APUF instances individually exhibit larger deviations from 50%. Further, we verified the uniformity (very close to 50%) of each challenge bit in our given set of CRPs, and also made sure that the training set and the test set do not have any overlap.

We observe that the prediction accuracy of the attack is between 55%-72%. This range is similar to that of the classical LR attack suggested in [NSJ+19] and adopted in [WMP+20] to model an (x,y) iPUF with an $x+y$-XOR-APUF, achieving the prediction accuracy of at most 75%. This problem of linearization attack caused by the effect of the interpose bit on the half of the bottom layer, has been claimed to be solved by the

[3] available at https://github.com/nils-wisiol/pypuf/tree/2020-split-ipuf

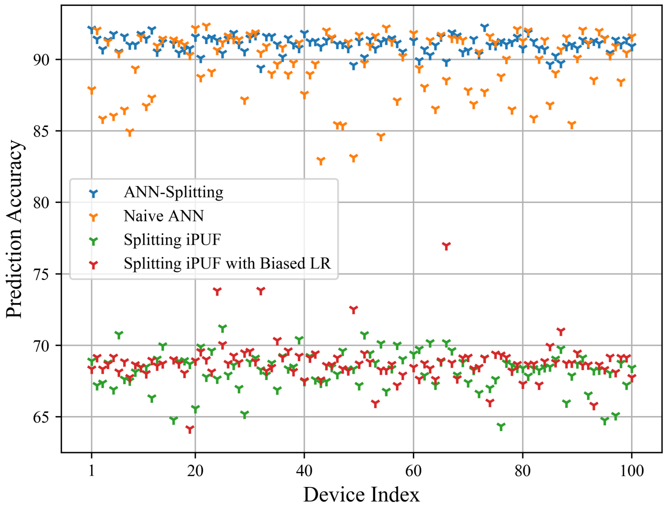


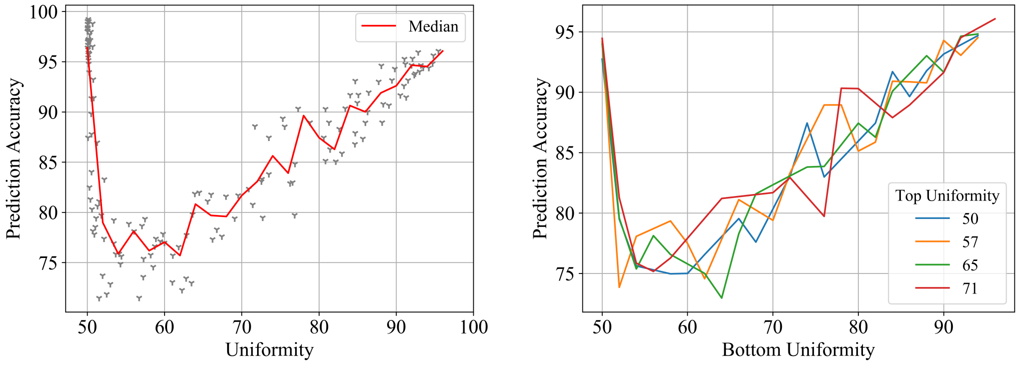
Figure 4: Maximum prediction accuracy of different attacks on 1 million CRP sets collected from the chosen $(1, 5)$ iPUF design implemented on a cluster of 100 FPGAs. Each attack has been repeated 10 times.

splitting iPUF attack. In contrast, our experimental analyses imply that this problem is not entirely eliminated, since non-uniformity of the underlying APUF instances as well as that of the iPUF can turn the highly-successful splitting iPUF attack into a naive LR modeling attack not profiting from its divide-and-conquer methods or CRP filtering.

In order to verify our results in the simulation domain and finding out the reason behind such an accuracy loss, we repeated the same attack on the same (1,5) iPUF design using simulated CRP sets for various amount of non-uniformity in the APUF instances of both top and down layers. To this end, we have adjusted a term so-called *bias* in the aforesaid library leading to non-uniform APUF models. For each selected bias, we collected 1 million noise-free CRPs, estimated the uniformity of the APUFs and iPUF response, and performed the splitting iPUF attack 10 times. The biased weights were simulated with the variance of 0.1, i.e., the individual uniformity of all APUF instances involved in each iPUF are mostly similar.

The corresponding results, shown in Figure 5a, first indicate that we cover the full range of uniformity between the ideal case (considered in the original splitting iPUF paper) and the worst case (which cannot fulfill the physical unclonability anymore). Second, it is shown that even with a slight bias $< 55\%$, the prediction accuracy falls below 80%. Naturally, the accuracy again gets back to the roll when the non-uniformity grows strongly to a side, but obviously such PUF constructions cannot be considered physically unclonable. We conduced another study by adjusting the bias of certain APUF instances. More precisely, for a specific non-uniformity of the top layer, we collected CRP sets and repeated the same attack for various non-uniformity of the bottom layer. Figure 5b depicts the results, illustrating that – in this case – the non-uniformity of the bottom layer is the dominant factor influencing the prediction accuracy of the attack.

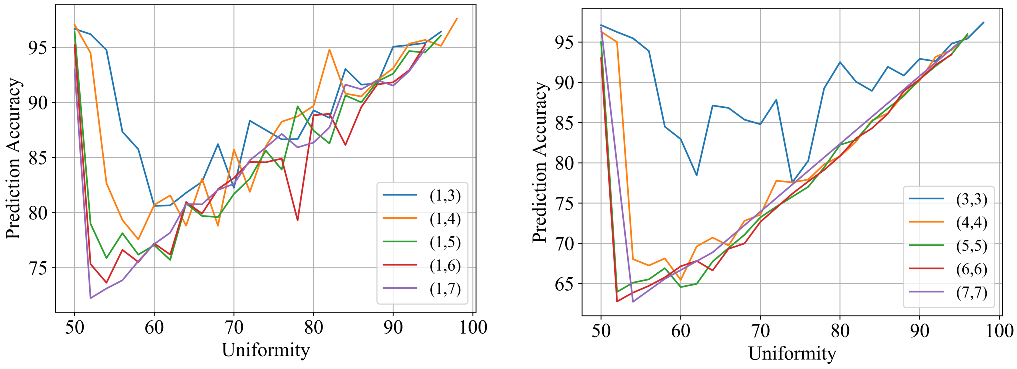
We further expended our analyses by studying other iPUF variants. The procedure explained above has been repeated on various 64-bit $(1, k)$ and (k, k) iPUF constructions. We again adjusted the bias parameter in the underlying APUF simulation models and considered the uniformity of the final response to observe the prediction accuracy of splitting iPUF attack. The results presented by Figure 6 reflect the same vision, i.e., by a



(a) Over uniformity of the final response

(b) For various uniformity in top and bottom layers

Figure 5: Prediction accuracy of splitting iPUF attack on $(1, 5)$ iPUF simulated with various range of uniformity. Each attack, repeated 10 times, is conducted on 1 million noise-free CRPs.



(a) Accuracy vs. uniformity for $(1, k)$ iPUF

(b) Accuracy vs. uniformity for (k, k) iPUF

Figure 6: Prediction accuracy of splitting iPUF attack on $(1, k)$ and (k, k) iPUF variants simulated with various range of uniformity. Each attack, repeated 10 times, is conducted on 1 million noise-free CRPs.

slight non-uniformity the prediction accuracy of the attack drops suddenly. Interestingly, as shown by the graphics, such an accuracy drop is sharper for larger iPUF variants, i.e., larger k.

We would like to emphasize that the original splitting iPUF attack is based on two instances of LR learner depending on the length of challenges (on top and bottom layers) in an iterative way. Features of our applied top and bottom LRs (which are similar to the original version) are summarized as: i) normal distribution of the initial weights $N(0, 1)$ in particulate the interpose bit from the previous layer, ii) convergence decimals of 2, and iii) no boundary on bias parameter. Note that the minimum applied iteration boundaries are also monotonous in both 64-bit and 65-bit learners with the value of 10 000. As stated in Section 2.3, there is a threshold for the accuracy of alternating phases of the splitting iPUF attack, which are updated at each iteration.

The results presented above are based on the original unaltered version of the attack. Below we theoretically discuss about the effect of bias on LR.

4.1.1 Effect of Bias on LR

We first should recall the concept of adding a regularization term to an error function to control the over-fitting problem through minimizing the total error function as shown in Equation (11). In the machine learning literature [Bis07, Har15], a particular choice of regularizer is known as weight decay, because in sequential learning algorithms, it encourages the weights to zero, unless supported by the data set.

$$E_D(W) + \lambda E_W(W) = -\ln p(C, r|W) + \lambda ||W||^2, \tag{11}$$

where λ shows the regularization coefficient. This coefficient controls the relative importance of the data-dependent error, shown as $E_D(W)$, and the regularization term $E_W(W)$. As it is also pointed out in [Bis07], the regularization provides a reasonable opportunity (but not as a definitive solution) for complex models to be trained on limited size of data sets to probably solve the over-fitting problem. Note that finding an optimal model, especially in complex cases (here the iPUF model), is shifted to finding an appropriate value of λ. Nevertheless, finding an optimal value of this parameter relies on various factors such as bias.

Below, we refer to two terms which – based on the regression literature – can affect the performance of the LR algorithm [Bis07, Har15]: 1) the "bias parameter" of the APUF linear model, and 2) the "bias term" involved in LR.

Bias parameter. The first term, also called offset β, is added to the APUF function, $f(W, C) = W^T \cdot \Phi + \beta$, where β – in contrast to weights w_i – is not limited to $[-1, 1]$. It is an extension to the delay model mentioned in Equation (1), where the bias parameter β has the role of determining the location of the decision boundary where $f(W, C) = 0$, i.e., where equal output probabilities occur. As one of the goals of LR is to minimize the error function (which is commonly supposed similar to cost/loss/utility function in a wrong way [Har15]), this bias parameter helps the sequential learning process to minimize the total error function in an efficient way. It can be interpreted that this parameter also places the learner on the right way of converging faster. Note that the bias of the training data sets is mostly ignored, or let say, it is supposed to be ideal $\beta = 0$ in the LR algorithms applied in attacks against PUFs, e.g., in splitting iPUF attack. One of the solutions to this problem is given in [JHTW14] where it is suggested to add the following transformation (log odds) to the bias β learned by the LR algorithm to tune it to reach the optimal value $\hat{\beta}$.

$$\widehat{\beta} = \beta + log\frac{p(C, r|W)}{1 - p(C, r|W)} - log\frac{p(C, \hat{r}|\hat{W})}{1 - p(C, \hat{r}|\hat{W})} \tag{12}$$

However, we show by our iPUF experiments in Section 4.1.2 that even if the attacker considers the bias of the final responses (the total uniformity of the iPUF), the optimal values of $\hat{W}$ cannot easily be found despite a slight improvement in the prediction accuracy. Here we should emphasize the complexity of the model of iPUF compared to other PUF architectures like single APUF or XOR-APUF when considering the effect of such parameters.

While the error function struggles to find a minimum distance from the optimal value in the complex models, it can also lead to severe over-fitting problems due to applying maximum likelihood or equivalently least-squares[4]. It should be searched for a suitable value of λ to reduce the over-fitting problem as much as possible. Simultaneously, the learner should also consider the bias to catch the most optimal value of $\hat{W}$.

[4] In required constrained models, the maximum likelihood or the estimation or the Bayesian distribution can be considered as the probability model as well. However, limiting the number of basis functions to avoid over-fitting has the side effect of limiting the flexibility of the model to capture the exciting and essential trends in the data set [Bis07].

Bias term. During regularization and minimizing the loss function, there is an effective parameter, so-called bias-variance of the learned weights, which should also be considered with respect to the type of the selected logistic function and the estimation approaches on the applied data sets [Bis07, Har15, KZ01]. The bias and variance of the fitting model generally have a relation with minimizing the loss function, which is a quantified measure to show how bad it is to get an error of a particular size or direction in the learning process. The loss function is also affected by the negative consequences of inaccurate prediction [OS18]. For the sake of simplicity and giving a general view, Equation (13) shows the effect of bias in the context of a linear model for logistic function, although it would also be applicable to more complex functions with various relation forms (more details are given in Appendix B).

$$\text{expected loss} \ \propto \ \text{bias} \propto \ \text{variance} \propto \ \text{noise} \tag{13}$$

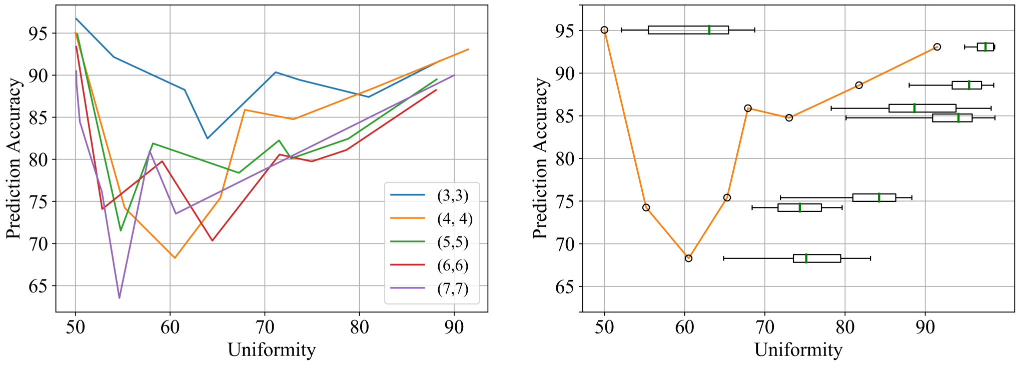
As mentioned in [Bis07], there is a trade-off between bias and variance, in which flexible models have low bias and high variance while relatively rigid models have high bias and low variance. Consequently, the regularization of the error functions λ affects minimizing bias and variance of the fitting model to provide the minimum error functions. As a solution regarding regularization, we additionally apply ANNs (given in Section 4.2), which consider the bias and the regularization in an efficient manner, while LR algorithm seems to be more sensitive to these biases.

Note that a proper probability model and a proper function for the decision should be chosen depending on the restriction hyperparameters and the effect of the noise and variance. In contrast to what is mentioned in [NSJ+19], there is no fixed rule implying that modeling approaches or supervised learners like LR or CMA-ES are always more capable or less capable of learning PUFs with different architectures since each learner has pros and cons. As we illustrated here, it can also be dependent on the CRP data set that gives us the related information.

4.1.2 Optimizations

In order to enhance the prediction accuracy of the attack, we have endeavored several approaches related to either the learner's features or the splitting iPUF attack algorithm. One of the first parameters, that we have considered in the underlying LR learner, is the interference of bias parameter as mentioned in Section 4.1.1. Although this term was defined in the lower layers of the applied library, it was not employed in various studies like splitting iPUF attack. More precisely, in the LR attack there is an opportunity to define the bias (non-uniformity) of the underlying PUF, for example the bias in the response of an APUF. If the attacker enters the bias of the top layer (which is supposed inaccessible) as well as the bottom layer, then this optimization can mitigate the non-uniformity. Note that adjusting the bias parameter can moderately improve the reduced accuracy in the case of iPUF model as a complex architecture but is not able to entirely avoid it. Further, the attacker does not have any chance to observe the uniformity of the top layer which creates the most important issue in the split learning of domino architecture of iPUF. Nevertheless, as a result of these modifications, we have observed a slight improvement in the prediction accuracy of the attacks. As shown in Figure 4, even the biased LR is not always able to compensate the accuracy loss except in a few data sets where the modified learner achieves the accuracy of at most 77%.

Due to the effect of the bias-variance trade-off in LR, shortening the sample sizes or weighting rare data can also be considered as a potential solution to the over-fitting problem [JLS13, Bis07, KZ01]. However, small sample sizes and classifying the balanced inputs to feed the input layer are the common ML techniques that have been used through pre-processing as well [HMOR94].



(a) Accuracy vs. uniformity for (k, k) iPUF (b) Accuracy vs. uniformity for $(4, 4)$ iPUF

Figure 7: Prediction accuracy of splitting iPUF attack on (k, k) iPUF variants simulated with various range of uniformity for each individual APUF. Each attack, repeated 10-20 times, is conducted on noise-free CRPs.

Our next option is to adjust the batch sizes and the applied number of iterations in LR. We have tried various batch sizes with small steps; however, this again did not help to compensate the accuracy loss. In some cases, considering the minimum batch sizes, the gradient descent does not pass the normal converging process and sticks far from finding a local or global minimum derivation point, in which the accuracy consequently falls to the worst case 50%, i.e., randomly guessing the PUF's output.

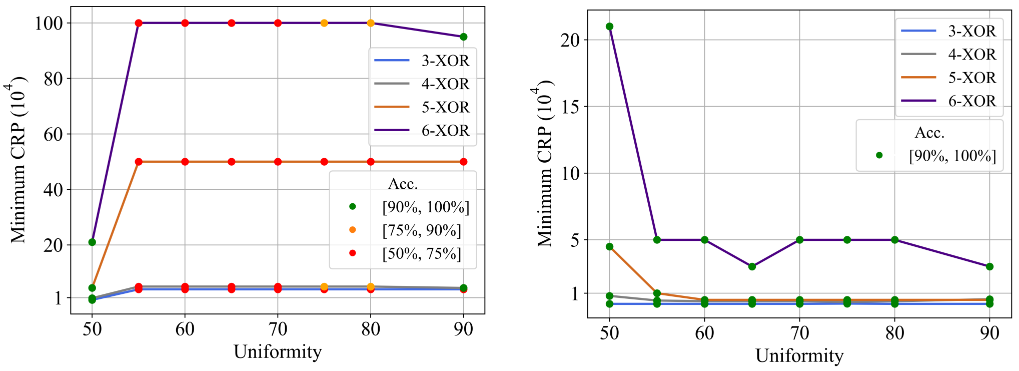
Moreover, the minimum number of required iterations for each LR has a minor impact on our desired output. One of the trivial solutions can be enlarging the applied input data set size, which here means an extensive CRP-set size. To cope with this, we increased the number of CRPs to 5 million and 10 million. We further collected similar number of CRPs from 10 devices of our FPGA cluster (the same design chosen in Section 3.2). However, we observed the same or even lower prediction accuracy in the result of the splitting iPUF attack.

Based on our experiments, the unique effective optimization is to deviate from uniform random selection of the interpose bit during the first phase of the splitting iPUF attack. Although it is recommended to keep a uniform distribution for the challenge bits and uniformly choose the initial value of the interpose bit at random, a non-uniform guessing of the interpose bit can relatively compensate for the accuracy loss and help to find the authentic model in several cases. We should mention that 1) this compensation irregularly moves the low prediction accuracy of 70% to above 80%, which is still far from a highly-accurate prediction model, and 2) in a realistic scenario the attacker does not have a chance to know the actual distribution of the interpose bit.

In order to have a fair comparison, we tried all above optimization options on the simulated CRP sets with a slight non-uniformity of 52%. In short, we observed that different iPUF variants, which are claimed to be breakable by splitting iPUF attack, cannot still be accurately modeled when the uniformity slightly deviates from its ideal value. To verify our experimental setup and the collected CRP sets, we successfully conduced classical LR attacks on each non-uniform XOR-APUF of the top and bottom layers independently (of course, knowing the output of the top layer).

4.1.3 Generalization

In this section, we extend our investigations by considering the effect of bias on two other case studies. In the simulation-based experiments shown in Figure 7b and Figure 7a,



(a) Minimum CRP vs. uniformity for k-XOR-APUF with unbiased LR

(b) Minimum CRP vs. uniformity for k-XOR-APUF with biased LR

Figure 8: The performance of two types of LR attack on k-XOR-APUFs in presence of bias. (a) unbiased LR, and (b) biased LR. Each attack, repeated 20 times, is conducted on noise-free CRPs.

we considered an approximately similar added bias (with small variance) for all APUF instances of an iPUF. Here, we first study other cases where every APUF has its individual uniformity in a range of $50\% - 95\%$, no necessarily the same as other APUFs in an iPUF instance. In the second case study, we examine the effect of bias on the efficiency of LR targeting XOR-APUFs, which are also involved in many common strong PUF constructions.

Various APUF Uniformity. In order to consider scenarios more close to real-world conditions, we investigate the behavior of the original splitting iPUF attack when each APUF has a different uniformity. We should highlight that this scenario is already the case for our practical experimental; as given in Table 1 and in Figure 3b, the APUFs have a wide range of uniformity. Here, we just examine the same scenario for different iPUF variants in the simulation domain. For the sake of consistency with Figure 5 and Figure 6, we present the results also based on the final uniformity (in the same range of $50\% - 95\%$). Therefore, the individual APUF's uniformity may strongly deviate from the center, which is actually not far from reality, as shown in Table 1. For each given final uniformity, we have examined more than 10 000 cases to find a combination of APUFs with different uniformity, and selected those with closest uniformity to 50%. The results shown in Figure 7a imply the same conclusion, i.e., the prediction accuracy decreases by non-uniformity of the final response. In order to give a more detailed view, Figure 7b presents the range of uniformity of individual APUFs for each final uniformity considered for $(4, 4)$ iPUF.

XOR-APUF. The latter case to generalize our observation is to investigate the effect of non-uniformity on XOR-APUFs. As given in Section 2.1, XOR-APUFs can be modeled easier compared to iPUFs. Therefore, the supervised learner succeeds more straightforwardly to achieve a high prediction accuracy with lower costs such as running time and the required number of CRPs. As shown in [TB15], when attacking k-XOR-APUFs, the minimum required CRPs to achieve a high accuracy and a reasonable success rate scale with k. Most of the pure ML attacks ignore the bias, and consider an ideal situation for the uniformity of all APUFs, like a range of $50 - 51\%$. In order to fill this gap, we ran several simulations for various k-XOR-APUFs $3 \leq k \leq 6$, where each APUF encounters a different bias, and conducted two types of LR and observed the minimum number of CRPs and the maximum prediction accuracy. In addition to the original so-called unbiased LR (from the employed library), where the bias is ignored, we considered the case in which

Table 2: Simulation results for the biased and unbiased LR for uniform and noise-free XOR-PUFs. For each instance, the attacks have been repeated 10 times.

k	**APUFs Uniformity**	**Final Uniformity**	**CRP size**		**Prediction Accuracy**	
			unbiased LR	biased LR	unbiased LR	biased LR
4	[50, 50, 50, 50]	50.00	15 000	15 000	96.07	96.10
	[45, 61, 64, 62]	50.09	50 000	10 000	82.11	96.75
	[59, 45, 71, 62]	50.28	50 000	7 500	81.91	96.88
5	[50, 50, 50, 50, 50]	50.00	50 000	50 000	92.43	92.09
	[43, 41, 47, 34, 30]	50.09	50 000	50 000	50.91	96.54
	[66, 57, 28, 32, 76]	50.31	50 000	45 000	52.02	96.87

the LR tries to learn the bias as well, referred to as biased LR. As shown in Figure 8a, the required number of CRPs for the unbiased LR highly increases when the uniformity deviates from 50%. In some cases, the prediction accuracy cannot even reach more than 75%. On the other hand, a lower number of CRPs is required in the biased LR while maintaining the prediction accuracy above 90%, and even less CRPs are required when observing stronger non-uniformity. The reason behind this reduction is that a biased CRP data set gives more information to the learner to model the underlying PUF faster. Note that the bias-variance trade-off of the learned weights should also be considered in the learner's decision stage.

We also investigate special cases where different XOR-APUFs exhibit almost ideal uniformity of 50% while the individual APUFs have various uniformity. For this experiment, we compare three different instances of 4-XOR-APUF (resp. 5-XOR-APUF), one of which with ideally-uniform APUFs. The results, depicted in Table 2, show even if the final uniformity is 50%, non-uniform individual APUFs can heavily decrease the prediction accuracy. Since it is not possible for the attacker to estimate the individual uniformity of APUF instances, the learner should be guided to consider the bias like the biased LR.

4.2 ANN and its Adoption to Splitting iPUF Attack

Since the splitting iPUF attack does not achieve an adequate prediction accuracy when the targeted PUF is slightly biased, our attention goes to the other learners like ANN, which is also embedded in the aforesaid pypuf library. Here, we show the result of the ANN attacks conducted to the same real data sets applied in Section 4.1, i.e., the chosen implementation of the $(1, 5)$ iPUF realized on 100 FPGA devices. In addition, we propose to substitute the LR learner of the splitting iPUF attack with an MLP to achieve a higher prediction accuracy in the presence of non-uniformity. Our proposed method can indeed be generalized by substituting the LR with other learners like Decision Tree (DT), which is supposed to be not as sensitive as LR to the data set's bias. Accordingly, we refer to the pure ANN attacks by "naive ANN" and our modified variant by "ANN-splitting attack".

Before illustrating the results, a description of the learner's parameters can give a detailed picture of the conducted process. The underlying MLP applies Rectified Linear Unit (ReLU) as the activation function and the Adaptive Moment Optimization (Adam), which combines the RMSProp and momentum. Adam computes individual adaptive learning rates from the first and second moments of the gradients to update the parameters [KB15]. In other words, this optimizer can deal efficiently with the noisy data sets by taking an average of the gradient distance of the learned weights. Since the learning parameters and batch sizes can be varied depending on the size of the training data set and the amount of the applied bias, we considered a wide range of learning rates (0.001-0.01) to have smaller step sizes and find an optimum point in the learning process. Further, we set the number of employed layers to be not more than 3 with (64, 64, 64) nodes. Note that we apply different learning rates and tolerance for the top and bottom layers in our ANN-splitting attack, e.g., the top and bottom layers use 0.001% and 0.003% learning rates, respectively. Regarding the effect of these parameters on the attack's running time, it is worth mentioning that each layer's patience term is also different. In other words, if there were a consecutive number of epochs that have low-change of loss compared to the threshold defined by the tolerance term, then the attack stops.

The results of these two ANN-based attacks on our non-uniform actual data sets are included in Figure 4 as well. As shown, the prediction accuracy of the naive ANN attack is in a range of 83% to 93%. This difference can be interpreted by the variation of the uniformity of APUF instances, since identical random initial weights and learning parameters have been applied for all CRP sets. On the other hand, by substituting the LR learner of the splitting iPUF attack with an ANN, we achieve a more stable learner's behavior leading to a higher and narrow prediction accuracy of 89%-93%. Note that in our ANN-splitting attack, we have not changed any other attack settings except replacing the learner. Although we discussed on the effect of different optimizations in Section 4.1.2, we stayed with the original settings; we have even kept a uniform distribution for the randomly-selected initial value of the interpose bit.

In short, in entire attacks conducted on all 100 devices, the naive ANN achieves a higher prediction accuracy compared to the original splitting iPUF attack, while it has less accuracy in approximately half of the devices when compared with our adopted ANN-splitting attack. We should highlight that for the results shown in Figure 4, each attack has been repeated 10 times with different initial weights, but the same initial weights have been considered for each instance of the attack on all 100 devices. Finally, the highest achieved accuracy is reported for each device.

Table 3 shows a summary of the conducted attacks. In addition to the design chosen in Section 3.2, we repeated our entire analyses on the second best design with the random placement strategy. In Table 3, we report the average prediction accuracy of each attack (repeated 10 times on each device). It can be seen that our ANN-splitting attack not only leads to a highest prediction accuracy, but also runs considerably faster compared to the naive ANN. This short running time is due to separating the learning process of each XOR-APUF layer, thereby giving the learner a chance to converge efficiently to the proper model in the back-and-forth process. As a side note, in order to handle all aforesaid extensive experiments in a reasonable time, we made use of a machine with 96 CPU cores with maximum clock frequency of 2.3 GHz and 256 GB of memory.

We conclude that when dealing with non-uniform complex PUF constructions, the divide-and-conquer approach of the splitting iPUF attack is a helpful technique that guides the learner to catch the optimum point faster compared to the naive ANN and classical LR attacks. However, referred to Section 4.1.1, in the underlying LR of the first divided-and-conquer phase of splitting iPUF attack, the error function shows a significant distance to the real responses, where the learner should meet a temporary accuracy threshold of less than 90%. Compared to LR, ANN has a higher chance to achieve a minimum distance due to the higher capability of the network and the involvement of regularization. Then, the obtained model is improved skillfully in the next phases of such a divided-and-conquer scheme in presence of non-uniformity. As a result, the attacks containing a precise model of the data sets (e.g., the iPUF architecture), instead of black-box attacks, enjoy a higher ability to more precisely learn the PUFs behavior.

5 Uniqueness and Reliability

Besides uniformity, other PUF metrics such as reliability and uniqueness play a critical role in the robustness of PUF constructions against ML attacks. Under the settings of this work, as mentioned in Section 2.4, the uniqueness shows the difference of outputs

Table 3: Average of prediction accuracy of attacks on $(1, 5)$ iPUF using real data sets collected from two designs with random placement implemented on 100 FPGA devices. Each attack has been repeated 10 times. The numbers written in parentheses indicate the average run time of a single attack.

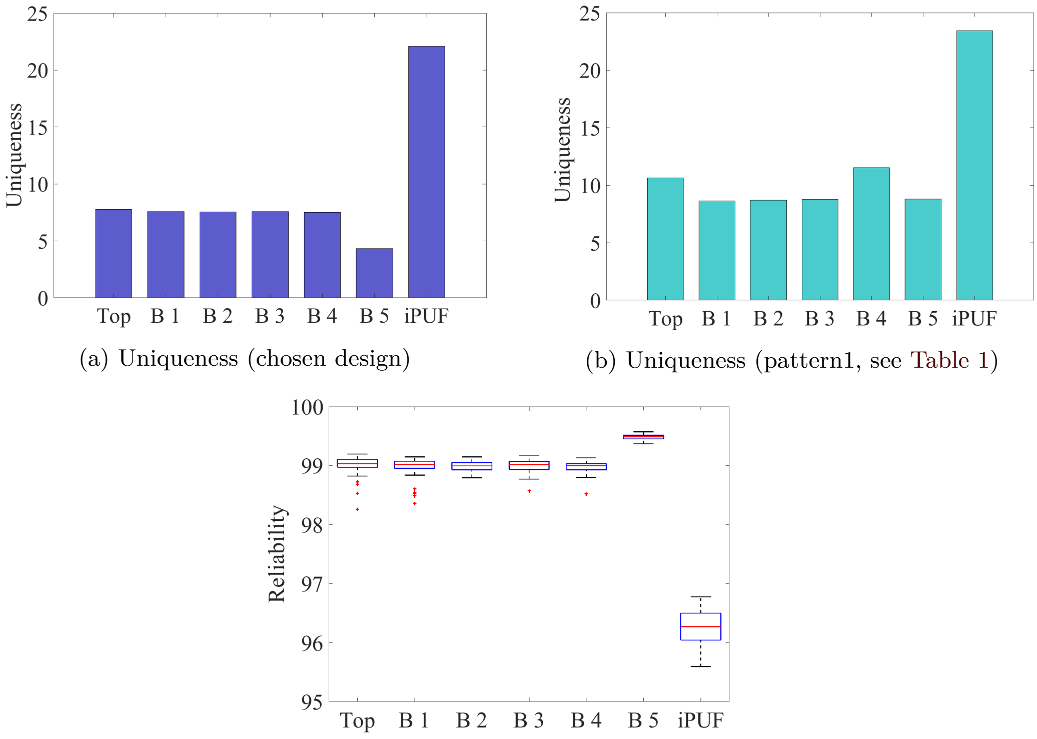
Design	**Prediction Accuracy (Time)**		
	Splitting iPUF	Naive ANN	ANN-Splitting
rand1	68.10% (11 min)	87.07% (42 min)	92.56% (14 min)
rand2	71.07% (11 min)	87.54% (42 min)	93.06% (15 min)

(responses) belonging to one PUF instance implemented on multiple devices. In other words, uniqueness measures the diversity of physical characteristics, i.e., an estimation of average dissimilarity in challenge-response behavior of PUF instances for the same given challenge. In an ideal case, the user expects to obtain various unpredictable responses with 0.5 or 50% uniqueness. It means that if two PUFs are totally containing unique and random responses for similar challenges, each of them should have a half of the other's responses. This parameter can be closely interpreted as the inter-(Hamming) distance of PUF architectures. Referred to our findings and as it is pointed out in [SNCM16], the non-uniformity of CRP data sets can affect the distribution of Hamming distance of the PUF responses. For instance, if a PUF design has a strong non-uniformity (bias) in its output, then the second chip with the same design and similar non-uniformity encounters with small variation in output. Therefore, PUF implementations face difficulties to achieve an ideal Hamming distance among several instances besides having an ideal uniformity.

In this section, we present the results of the evaluation of the $(1, 5)$ iPUF design chosen in Section 3.2 in terms of uniqueness and reliability. Using the data sets we collected from all 100 devices, we examined the uniqueness of the chosen implementation. The corresponding results shown in Figure 9a imply a very low uniqueness for the APUF instances, far from an ideal situation (less than 10%). Although it is affected by non-uniformity, the low uniqueness in FPGA implementation of delay-based PUFs is usually due to the relatively-identical architectures of the platform. More precisely, it is due to the fact that the difference between the delay lines (e.g., of an APUF switch stage) is not heavily affected by the process variation, rather by the fixed routing resources. Apart from these facts and observations, we report a considerably-higher uniqueness for the overall iPUF architecture compared to the underlying APUF instances, i.e., 23% versus at most 8%. We expect this to be the result of the underlying composite architecture of the iPUF.

It is worth to recall the recommendations made in [NSJ$^+$19] with respect to the placement strategy, which we extensively discussed in Section 3.1. In addition to our chosen design, we have taken one of the recommended pattern-based designs, indicated as "pattern1" in Table 1 showing maximum uniformity for each APUF instance among all 11 pattern-based designs. We repeated the same procedure, i.e., implementing the design on all 100 FPGAs, collecting 1 million CRPs, and examining the uniqueness. As depicted in Figure 9b, the recommended design shows a slight improvement in the uniqueness of APUF instances. However, this improvement is not scaled for the uniqueness of the iPUF. We should point out that we have examined several other designs with different placement strategies, in none of them we observed a uniqueness higher than 24% for the iPUF.

As a result of this low uniqueness, it is expected that these constructions encounter physical clonability issues. In order to investigate this, we have conducted another experiment. Considering the result of our ANN-splitting attack on all 100 devices, we have selected the best-trained model in terms of the prediction accuracy of 92.5%. Using this model, we have tried to predict the response of other 99 devices, of course implementing the same design. To this end, we applied a test set of 50 k CRPs to the chosen model and



(a) Uniqueness (chosen design)

(b) Uniqueness (pattern1, see Table 1)

(c) Average reliability

Figure 9: Uniqueness and average reliability of the chosen $(1,5)$ iPUF design, based on the CRPs collected from 100 devices.

to all 99 devices. As shown by Figure 10, this experiment leads to the prediction accuracy of at most 83%, which is approximately in line with the observed uniqueness of 23%. In short, due to the lower uniqueness of APUF instances, we expected to achieve a higher prediction accuracy when we port the model of one device to the others. However, the iPUF's composite architecture, particularly the interpose bit given to the middle of the bottom layer, improves but not fully compensates for the physical clonability.

Reliability

Researchers are commonly hesitant whether APUFs on FPGA can achieve a high reliability [WMP$^+$20, SNCM16]. By means of Figure 9c, which shows the average reliability of the entire components of the underlying iPUF, we present our observations toward a highly-reliable situation. Surprisingly, the implementation enjoys a high reliability, relatively close to ideal. As given in Section 3.1, we tried to limit the environmental effects/noise on our setup. Despite the high reliability of our case study, as stated in Section 4.1, we have conducted all our attacks on more-reliable CRP sets by applying majority voting after repeating each challenge 11 times.

6 Discussions

The primary goal of ML attacks targeting constructions involving APUFs, is to learn the weights of the underlying model (see Equation (1)) in a way that the predicted model is able to behave error-less compared to the actual PUF. One of the techniques, which

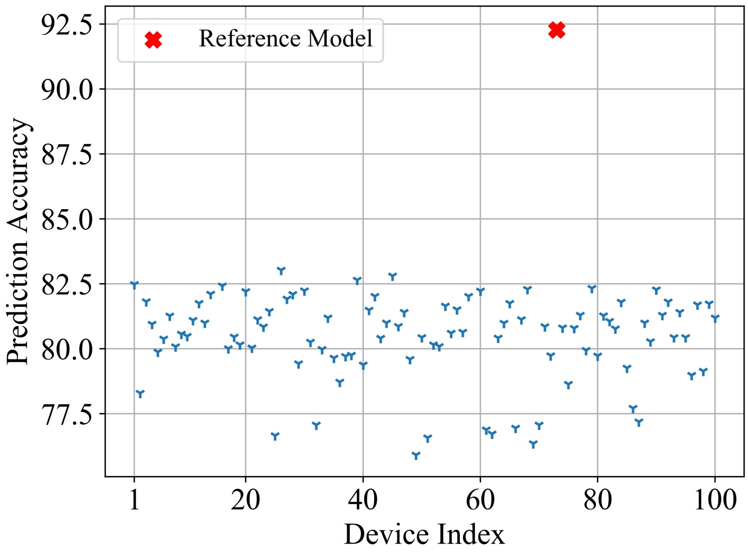


Figure 10: Prediction accuracy of a selected reference model applied on 99 sets of CRPs collected from 99 devices. The reference model has been selected as the one that achieved the highest prediction accuracy as a result of ANN-splitting attack.

might be helpful during the learning process, is to observe the weights correlations and the delay differences of various applied APUFs in a composite architecture, as it has been done in [DV13, Del17, TAB20, Bec15, WMP+20]. To be more precise, here weights correlation refers to the correlation between the weights used to generate the simulated CRPs and those revealed by the learning process. Of course, it is just applicable in the simulation domain. During our simulation-based investigations, we have checked out the weights correlation for individual APUFs in both iPUF layers. As a result, those splitting iPUF attacks which reach the prediction accuracy of around 80%, enjoy a proper weights correlation, i.e., 1 or -1; however, the accuracy does not improve through more iterations of the attack. Naturally, it is of great interest if this shortcoming is addressed and that such attacks are optimized to achieve a higher success rate. Note that choosing the initial values of the interpose bit from a non-uniform distribution (discussed in Section 4.1.2) has also contributed to boost this correlation, when dealing with our non-ideally uniform case studies. This point leads us to recall the discussion given in [WMP+20] implying that a low-prediction accuracy does not prove the PUF's resilience against LR attacks. We would like to expand this statement by giving another perspective that high prediction accuracy and success rate of ML attacks in the simulation domain does not mean the absolute breakdown of the underlying construction in the practical domain. As investigated here, we showed that there are more parameters such as uniformity, which have a significant impact on the ML attack's prediction accuracy and success rate.

In order to have a practically-secure PUF design, two solutions are commonly given by the the researchers [WMP+20, NSJ+19, RSS+10]: 1) expanding the PUF challenge size and/or 2) increasing the number of applied APUFs in the architecture. The first solution is usually preferred, since it is assumed that increasing the number of APUF components can cause a sharp reduction in its reliability. In our practical experiments, each 64-bit APUF enjoys a high reliability of around 99%. This led to a stable composite construction $(1, 5)$ iPUF as shown in Figure 9c. Based on the statements and estimations given in [NSJ+19], for a larger variant (1,12) iPUF implemented on our setup, we predict the reliability to be in a range of 88.6%-94.1%. Of course, these results are based on our

FPGA-based setup, and cannot be directly mapped to any other setup.

In the end, it should be mentioned that the conclusion of XOR APUF non-uniformity should not be taken in a wrong interpretation that the iPUF structure is similar to XOR APUF. As comprehensively discussed in [WMP+20], the modeling of x+y-XOR APUF can not be chosen for (x, y) iPUF because it is not reflecting all characteristics of iPUF. The key point of affecting the interpose bit on the half of the bottom layer's weights plays a vital role in making iPUF distinguished compared to other PUF structures.

Practical View

Here, we would like to give our practice-oriented vision on ML attacks, especially splitting iPUF and ANN. For instance, based on the analyses given in [WMP+20], the splitting iPUF attack on the (1,9) iPUF needs 750 million ideally-uniform CRPs. Collecting such a CRP set takes around 10 days using our setup (of course optimizations are still possible). Since the splitting iPUF attack on noise-free uniform data sets with that size took around 8 weeks reaching a success rate of 26%, we expect that our adopted ANN-splitting attack on such a construction takes even longer when the CRP sets are experimentally collected from a non-uniform construction (in total 2-3 months including CRP collection). Therefore, successfully conducting a pure ML attack on large and robust APUF-based architectures sounds challenging from a practical perspective. In other words, we somehow confirm the statements made in [WMP+20, NSJ+19] regarding the infeasibility of the attacks and collection of required CRP sets, i.e., the required time increases with polynomial complexity. Therefore, such attacks are possible but not practical on large iPUF variants. Note that side-channel information like reliability or power consumption can mitigate the difficulties of ML attacks in practice. They usually are able to turn that difficulty to a linear relationship between the required CRPs and the number of APUF instances involved in a composed architecture. For the sake of repeatability, a set of 1 million CRPs collected from an iPUF implementation in our setup is accessible through the authors' webpage.

7 Conclusion

PUF metrics have been thoroughly studied by several researchers. In this work, we have highlighted the crucial role of such metrics in the success of corresponding modeling attacks. Contrary to a common assumption that implementation defects can ease the attacks, using several practical and simulation-based analyses, we have shown that a slight non-uniformity in the output of the PUF with a complex architecture can turn a theoretically-successful ML attack into a model with moderate prediction accuracy.

To this end, we focused on the recently introduced splitting iPUF attack and presented its difficulties when targeting an iPUF variant with some bias in practice. We have also observed that by deviating from the ideal uniformity, the prediction accuracy of the attack on large iPUF constructions drops more sharply compared to that of smaller variants. To the best of our knowledge, this article is the first pointing out these issues supported by extensive experiments performed on a large-scale FPGA platform. We further have proposed some modifications on the splitting iPUF attack to tolerate the non-uniformity. Making use of other learners like DT, SVM, and CMA-ES instead of LR and ANN (also in the alternative form of splitting iPUF attack) may lead to better enhancements when dealing with biased constructions.

In short, we believe that based on our results, any attack vector developed in the future should be examined under non-ideal situations as well, e.g., facing bias originating from implementation defects. Otherwise, an unconscious conclusion may be obtained about the resilience/vulnerability of the underlying PUF construction against ML classifiers like LR. To avoid the inconsistency between the simulation and real-world conditions, we should pay attention to the complexity of PUF architectures, when selecting proper modeling attacks.

Acknowledgments

The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA.

References

[AM20] Anita Aghaie and Amir Moradi. TI-PUF: toward side-channel resistant physical unclonable functions. *IEEE Trans. Inf. Forensics Secur.*, 15:3470–3481, 2020.

[Bec15] Georg T. Becker. The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs. In *Cryptographic Hardware and Embedded Systems - CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 535–555. Springer, 2015.

[Bis07] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition.* Information science and statistics. Springer, 2007.

[BK14] Georg T. Becker and Raghavan Kumar. Active and Passive Side-Channel Attacks on Delay Based PUF Designs. *IACR Cryptol. ePrint Arch.*, 2014:287, 2014.

[CCL$^+$11] Qingqing Chen, György Csaba, Paolo Lugli, Ulf Schlichtmann, and Ulrich Rührmair. The Bistable Ring PUF: A new architecture for strong Physical Unclonable Functions. In *IEEE International Symposium on Hardware-Oriented Security and Trust - HOST 2011*, pages 134–141. IEEE Computer Society, 2011.

[CDS18] Baibhab Chatterjee, Debayan Das, and Shreyas Sen. RF-PUF: IoT security enhancement through authentication of wireless nodes using in-situ machine learning. In *IEEE International Symposium on Hardware Oriented Security and Trust - HOST 2018*, pages 205–208. IEEE Computer Society, 2018.

[CM91] Gauss M Cordeiro and Peter McCullagh. Bias correction in generalized linear models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(3):629–643, 1991.

[DBNT19] Samir Ben Dodo, Rajendra Bishnoi, Sarath Mohanachandran Nair, and Mehdi Baradaran Tahoori. A Spintronics Memory PUF for Resilience Against Cloning Counterfeit. *IEEE Trans. Very Large Scale Integr. Syst.*, 27(11):2511–2522, 2019.

[Del17] Jeroen Delvaux. *Security Analysis of PUF-based Key Generation and Entity Authentication.* PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 2017.

[Del19] Jeroen Delvaux. Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF-FSMs. *IEEE Trans. Inf. Forensics Secur.*, 14(8):2043–2058, 2019.

[DGSV14] Jeroen Delvaux, Dawu Gu, Dries Schellekens, and Ingrid Verbauwhede. Secure Lightweight Entity Authentication with Strong PUFs: Mission Impossible? In *Cryptographic Hardware and Embedded Systems - CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 451–475. Springer, 2014.

[DGSV15] Jeroen Delvaux, Dawu Gu, Dries Schellekens, and Ingrid Verbauwhede. Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 34(6):889–902, 2015.

[DGV$^+$16] Jeroen Delvaux, Dawu Gu, Ingrid Verbauwhede, Matthias Hiller, and Meng-Day (Mandel) Yu. Efficient Fuzzy Extraction of PUF-Induced Secrets: Theory and Applications. In *Cryptographic Hardware and Embedded Systems - CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 412–431. Springer, 2016.

[DV13] Jeroen Delvaux and Ingrid Verbauwhede. Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise. In *IEEE International Symposium on Hardware-Oriented Security and Trust - HOST 2013*, pages 137–142. IEEE Computer Society, 2013.

[DV14] Jeroen Delvaux and Ingrid Verbauwhede. Key-recovery attacks on various RO PUF constructions via helper data manipulation. In *Design, Automation & Test in Europe Conference & Exhibition - DATE 2014*, pages 1–6, 2014.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[GCvDD02] Blaise Gassend, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Conference on Computer and Communications Security - CCS 2002*, pages 148–160. ACM, 2002.

[GFS19] Fatemeh Ganji, Domenic Forte, and Jean-Pierre Seifert. PUFmeter a Property Testing Tool for Assessing the Robustness of Physically Unclonable Functions to Machine Learning Attacks, 2019.

[GTS16] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. PAC learning of arbiter PUFs. *J. Cryptogr. Eng.*, 6(3):249–258, 2016.

[GTS18] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. A Fourier Analysis Based Attack Against Physically Unclonable Functions. In *Financial Cryptography and Data Security - FC 2018*, volume 10957 of *Lecture Notes in Computer Science*, pages 310–328. Springer, 2018.

[GTS$^+$19] Fatemeh Ganji, Shahin Tajik, Pascal Stauss, Jean-Pierre Seifert, Domenic Forte, and Mark Tehranipoor. Rock'n'roll PUFs: Crafting Provably Secure PUFs from Less Secure Ones. In *Workshop on Security Proofs for Embedded Systems - PROOFS 2019*, volume 11 of *Kalpa Publications in Computing*, pages 33–48, 2019.

[Har15] F.E. Harrell. *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis.* Springer Series in Statistics. Springer International Publishing, 2015.

[HKM$^+$12] Anthony Van Herrewege, Stefan Katzenbeisser, Roel Maes, Roel Peeters, Ahmad-Reza Sadeghi, Ingrid Verbauwhede, and Christian Wachsmann. Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs. In *Financial Cryptography and Data Security - FC*

2012, volume 7397 of *Lecture Notes in Computer Science*, pages 374–389. Springer, 2012.

[HMOR94] Tim Hill, Leorey Marquez, Marcus O'Connor, and William Remus. Artificial neural network models for forecasting and decision making. *International Journal of Forecasting*, 10(1):5–15, 1994.

[HYKD14] Charles Herder, Meng-Day (Mandel) Yu, Farinaz Koushanfar, and Srinivas Devadas. Physical Unclonable Functions and Applications: A Tutorial. *Proceedings of the IEEE*, 102(8):1126–1141, 2014.

[HYKS10] Yohei Hori, Takahiro Yoshida, Toshihiro Katashita, and Akashi Satoh. Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs. In *Conference on Reconfigurable Computing and FPGAs - ReConFig 2010*, pages 298–303. IEEE Computer Society, 2010.

[JHTW14] Gareth James, Trevor Hastie, Robert Tibshirani, and Daniela Witten. *An Introduction to Statistical Learning: With Applications in R.* Springer, 2014.

[JLS13] David W. Hosmer Jr., Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression, 3rd Edition.* Wiley, 2013.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Conference on Learning Representations - ICLR 2015*, 2015.

[KKR$^+$12] Stefan Katzenbeisser, Ünal Koçabas, Vladimir Rozic, Ahmad-Reza Sadeghi, Ingrid Verbauwhede, and Christian Wachsmann. PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon. In *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 283–301. Springer, 2012.

[KZ01] Gary King and Langche Zeng. Logistic regression in rare events data. *Political Analysis*, 9(2):137âĂŞ163, 2001.

[LLG$^+$04] Jae W Lee, Daihyun Lim, Blaise Gassend, G Edward Suh, Marten Van Dijk, and Srinivas Devadas. A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications. In *Symposium on VLSI Circuits. Digest of Technical Papers*, pages 176–179. IEEE, 2004.

[MGS11] Abhranil Maiti, Vikash Gunreddy, and Patrick Schaumont. A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. *IACR Cryptol. ePrint Arch.*, 2011:657, 2011.

[MKD10] Mehrdad Majzoobi, Farinaz Koushanfar, and Srinivas Devadas. FPGA PUF using programmable delay lines. In *Workshop on Information Forensics and Security - WIFS 2010*, pages 1–6. IEEE, 2010.

[MKKD14] Mehrdad Majzoobi, Akshat Kharaya, Farinaz Koushanfar, and Srinivas Devadas. Automated Design, Implementation, and Evaluation of Arbiter-based PUF on FPGA using Programmable Delay Lines. *IACR Cryptology ePrint Archive*, 2014:639, 2014.

[MRK$^+$12] Mehrdad Majzoobi, Masoud Rostami, Farinaz Koushanfar, Dan S. Wallach, and Srinivas Devadas. Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching. In *Symposium on Security and Privacy Workshops 2012*, pages 33–44. IEEE Computer Society, 2012.

[MRV$^+$12] Roel Maes, Vladimir Rozic, Ingrid Verbauwhede, Patrick Koeberl, Erik van der Sluis, and Vincent van der Leest. Experimental evaluation of Physically Unclonable Functions in 65 nm CMOS. In *European Solid-State Circuit conference - ESSCIRC 2012*, pages 486–489. IEEE, 2012.

[MS11] Abhranil Maiti and Patrick Schaumont. Improved Ring Oscillator PUF: An FPGA-friendly Secure Primitive. *J. Cryptol.*, 24(2):375–397, 2011.

[MvdL14] Roel Maes and Vincent van der Leest. Countering the effects of silicon aging on SRAM PUFs. In *Symposium on Hardware-Oriented Security and Trust - HOST 2014*, pages 148–153. IEEE Computer Society, 2014.

[MYIS15] Takanori Machida, Dai Yamamoto, Mitsugu Iwamoto, and Kazuo Sakiyama. A New Arbiter PUF for Enhancing Unpredictability on FPGA. *The Scientific World Journal*, 2015:13, 2015.

[MYKP20] Saraju P. Mohanty, Venkata P. Yanambaka, Elias Kougianos, and Deepak Puthal. PUFchain: A Hardware-Assisted Blockchain for Sustainable Simultaneous Device and Data Security in the Internet of Everything (IoE). *IEEE Consumer Electron. Mag.*, 9(2):8–16, 2020.

[NJGS09] Szilard Nemes, Junmei Miao Jonasson, Anna Genell, and Gunnar Steineck. Bias in odds ratios by logistic regression modelling and sample size. *BMC Medical Research Methodology*, 9(1):56, 2009.

[NS14] Phuong Ha Nguyen and Durga Prasad Sahoo. Lightweight and Secure PUFs: A Survey (Invited Paper). In *Coneference on Security, Privacy, and Applied Cryptography Engineering - SPACE 2014*, volume 8804 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2014.

[NSJ$^+$19] Phuong Ha Nguyen, Durga Prasad Sahoo, Chenglu Jin, Kaleel Mahmood, Ulrich Rührmair, and Marten van Dijk. The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(4):243–290, 2019.

[NSMC14] Phuong Ha Nguyen, Durga Prasad Sahoo, Debdeep Mukhopadhyay, and Rajat Subhra Chakraborty. Cryptanalysis of Composite PUFs (Extended abstract-invited talk). In *Symposium on VLSI Design and Test - VDAT 2014*, pages 1–2. IEEE, 2014.

[OS18] B. O'Neill and S. Sanni. Profit optimisation for deterministic inventory systems with linear cost. *Computers and Industrial Engineering*, 122:303–317, 2018.

[Pos98] Reinhard Posch. Protecting Devices by Active Coating. *J. Univers. Comput. Sci.*, 4(7):652–668, 1998.

[PRTG02] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical One-Way Functions. *Science*, 297(5589):2026–2030, 2002.

[RH14] Ulrich Rührmair and Daniel E. Holcomb. PUFs at a Glance. In *Design, Automation & Test in Europe Conference & Exhibition - DATE 2014*, pages 1–6, 2014.

[RSS$^+$10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling Attacks on Physical Unclonable Functions. In *Computer and Communications Security - CCS 2010*, pages 237–249. ACM, 2010.

[RSS$^+$13] Ulrich Rührmair, Jan Sölter, Frank Sehnke, Xiaolin Xu, Ahmed Mahmoud, Vera Stoyanova, Gideon Dror, Jürgen Schmidhuber, Wayne Burleson, and Srinivas Devadas. PUF Modeling Attacks on Simulated and Silicon Data. *IEEE Trans. Inf. Forensics Security"*, 8(11):1876–1891, 2013.

[RXS$^+$14] Ulrich Rührmair, Xiaolin Xu, Jan Sölter, Ahmed Mahmoud, Mehrdad Majzoobi, Farinaz Koushanfar, and Wayne P. Burleson. Efficient Power and Timing Side Channels for Physical Unclonable Functions. In *Cryptographic Hardware and Embedded Systems - CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 476–492. Springer, 2014.

[SBC19] Pranesh Santikellur, Aritra Bhattacharyay, and Rajat Subhra Chakraborty. Deep Learning based Model Building Attacks on Arbiter PUF Compositions. *IACR Cryptol. ePrint Arch.*, 2019:566, 2019.

[SCM15] Durga Prasad Sahoo, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. Towards Ideal Arbiter PUF Design on Xilinx FPGA: A Practitioner's Perspective. In *Euromicro Conference on Digital System Design - DSD 2015*, pages 559–562. IEEE Computer Society, 2015.

[SD07] G. Edward Suh and Srinivas Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Design Automation Conference - DAC 2007*, pages 9–14. IEEE, 2007.

[SG18] Geert-Jan Schrijen and Cesare Garlati. Physical Unclonable Functions to the Rescue – A New Way to Establish Trust in Silicon. In *Embedded World 2018*, 2018.

[SH14] Dieter Schuster and Robert Hesselbarth. Evaluation of Bistable Ring PUFs Using Single Layer Neural Networks. In *Trust and Trustworthy Computing - TRUST 2014*, volume 8564 of *Lecture Notes in Computer Science*, pages 101–109. Springer, 2014.

[SMCN18] Durga Prasad Sahoo, Debdeep Mukhopadhyay, Rajat Subhra Chakraborty, and Phuong Ha Nguyen. A Multiplexer-Based Arbiter PUF Composition with Enhanced Reliability and Security. *IEEE Trans. Computers*, 67(3):403–417, 2018.

[SMKT06] Boris Skoric, Stefan Maubach, Tom A. M. Kevenaar, and Pim Tuyls. Information-theoretic analysis of coating PUFs. *IACR Cryptol. ePrint Arch.*, 2006:101, 2006.

[SNCM16] Durga Prasad Sahoo, Phuong Ha Nguyen, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. Architectural Bias: a Novel Statistical Metric to Evaluate Arbiter PUF Variants. *IACR Cryptol. ePrint Arch.*, 2016:57, 2016.

[TAB20] Johannes Tobisch, Anita Aghaie, and Georg T. Becker. Combining Optimization Objectives: New Machine-Learning Attacks on Strong PUFs. *IACR Cryptol. ePrint Arch.*, 2020:957, 2020.

[TB15] Johannes Tobisch and Georg T. Becker. On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation. In *RFIDsec 2015*, volume 9440 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2015.

[TDF$^+$17] Shahin Tajik, Enrico Dietz, Sven Frohmann, Helmar Dittrich, Dmitry Nedospasov, Clemens Helfmeier, Jean-Pierre Seifert, Christian Boit, and Heinz-Wilhelm Hübers. Photonic Side-Channel Analysis of Arbiter PUFs. *J. Cryptol.*, 30(2):550–571, 2017.

[WMP$^+$20] Nils Wisiol, Christopher Mühl, Niklas Pirnay, Phuong Ha Nguyen, Marian Margraf, Jean-Pierre Seifert, Marten van Dijk, and Ulrich Rührmair. Splitting the Interpose PUF: A Novel Modeling Attack Strategy. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):97–120, 2020.

[WP20] Nils Wisiol and Niklas Pirnay. Short Paper: XOR Arbiter PUFs Have Systematic Response Bias. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - FC 2020*, volume 12059 of *Lecture Notes in Computer Science*, pages 50–57. Springer, 2020.

[YHD$^+$16] Meng-Day (Mandel) Yu, Matthias Hiller, Jeroen Delvaux, Richard Sowell, Srinivas Devadas, and Ingrid Verbauwhede. A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication. *IEEE Trans. Multi-Scale Computing Systems*, 2(3):146–159, 2016.

[YMVD14] Meng-Day (Mandel) Yu, David M'Raïhi, Ingrid Verbauwhede, and Srinivas Devadas. A noise bifurcation architecture for linear additive physical functions. In *Symposium on Hardware-Oriented Security and Trust - HOST 2014*, pages 124–129. IEEE Computer Society, 2014.

[ZBX$^+$20] Jun Zhao, Weixin Bian, Deqin Xu, Biao Jie, Xintao Ding, Wen Zhou, and Hui Zhang. A Secure Biometrics and PUFs-Based Authentication Scheme With Key Agreement For Multi-Server Environments. *IEEE Access*, 8:45292–45303, 2020.

A Implementation Platform

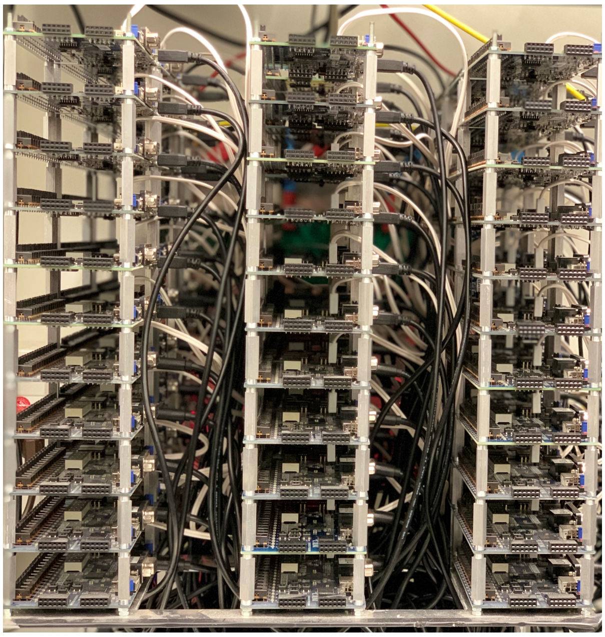


Figure 11: Our employed FPGA cluster, containing 100 Artix-7 FPGAs, on which various iPUF instances are implemented.

B Adopted Bias-Variance Relation

In the decision theory of regression, there are various problems like over-fitting, which leads us to consider a regularization in the estimation of the trained value $\hat{f}(W, C)$, corresponding to the real output r. Also, it leads us to calculate the minimum of the loss function in order to achieve the optimal prediction on our conditional distribution of $p(C, r|W)$ for the PUF function as mentioned before. The proper approaches to minimize the loss function can be chosen among different methods like minimizing least squares or maximum likelihood, which are applied on a determined data set M.

In the following, we present the adopted version of finding the bias-variance trade-off by minimizing the loss function regarding learning the APUF model, which is supposed to be learned as a linear model. For the sake of simplicity, we consider linear regression and square loss function in our example to give a general view to the reader by the aim of understanding the bias-variance trade-off and their relation to the expected loss function [FHT01, Bis07]. However, it is worth to mention that applying maximum likelihood estimation on the model parameters in the error function should not be supposed the same as minimizing the least square of the loss function in the regression area.

Here, we start with the conditional expectation of our function, $f(W, C)$, over data set M as

$$\mathbb{E}[f(W, C)|M] = f(W, C). \tag{14}$$

If we consider an independent noise term, shown by ϵ in our function, then the function can be shown like $y(W, C) = f(W, C) + \epsilon$. The conditional expectation of the new function is the same as what we have in Equation (14):

$$\mathbb{E}[y(W, C)] = \mathbb{E}[f(W, C)] + \mathbb{E}[\epsilon] = \mathbb{E}[f(W, C)], \text{ where } \mathbb{E}[\epsilon] = 0 \tag{15}$$

We should then find the expected loss function considering the above equations, the trained PUF function $\hat{f}(W,C)$, and variance Var as applied in the following equations. Hence, we have the loss function through minimizing square as follows [FHT01]:

$$
\begin{aligned}
&\mathbb{E}[L] = \mathbb{E}[\big(y(W,C) - \hat{f}(W,C)\big)^2] = \mathbb{E}[\big(f(W,C) + \epsilon - \hat{f}(W,C)\big)^2] \\
&= \mathbb{E}[\big(f(W,C) + \epsilon - \hat{f}(W,C) + \mathbb{E}[\hat{f}(W,C)] - \mathbb{E}[\hat{f}(W,C)]\big)^2] \\
&= \mathbb{E}[(f(W,C) - \mathbb{E}[\hat{f}(W,C)])^2] + \mathbb{E}[\epsilon^2] + \mathbb{E}[(\mathbb{E}[\hat{f}(W,C)] - \hat{f}(W,C))^2] \\
&+ 2\mathbb{E}[(f(W,C) - \mathbb{E}[\hat{f}(W,C)])\epsilon] + 2\mathbb{E}[\epsilon(\mathbb{E}[\hat{f}(W,C)] - \hat{f}(W,C))] \\
&+ 2\mathbb{E}[(\mathbb{E}[\hat{f}(W,C)] - \hat{f}(W,C))(f(W,C) - \mathbb{E}[\hat{f}(W,C)])] \\
&= (f(W,C) - \mathbb{E}[\hat{f}(W,C)])^2 + \mathbb{E}[\epsilon^2] + \mathbb{E}[(\mathbb{E}[\hat{f}(W,C)] - \hat{f}(W,C))^2] \\
&+ 2(f(W,C) - \mathbb{E}[\hat{f}(W,C)])\mathbb{E}[\epsilon] + 2\mathbb{E}[\epsilon]\mathbb{E}[\mathbb{E}[\hat{f}(W,C)] - \hat{f}(W,C)] \\
&+ 2\mathbb{E}[\mathbb{E}[\hat{f}(W,C)] - \hat{f}(W,C)](f(W,C) - \mathbb{E}[\hat{f}(W,C)]) \\
&= (f(W,C) - \mathbb{E}[\hat{f}(W,C)])^2 + \mathbb{E}[\epsilon^2] + \mathbb{E}[(\mathbb{E}[\hat{f}(W,C)] - \hat{f}(W,C))^2] \\
&= (f(W,C) - \mathbb{E}[\hat{f}(W,C)])^2 + \mathrm{Var}[\epsilon] + \mathrm{Var}[\hat{f}(W,C)] \\
&= \mathrm{Bias}[\hat{f}(W,C)]^2 + \mathrm{Var}[\epsilon] + \mathrm{Var}[\hat{f}(W,C)] \\
&= \mathrm{Bias}[\hat{f}(W,C)]^2 + \mathrm{Var}[\hat{f}(W,C)] + \mathrm{noise}\,. \qquad (16)
\end{aligned}
$$

As shown, the expected loss is related to the bias and variance of the trained model (resp. learned weights in our PUF application). In other words, the loss function is split into the sum of (squared) bias, variance, and a constant noise. Note that there is a trade-off between bias and variance as well so that the bias term represents the extent to which the average prediction over all data sets differs from the desired regression function [Bis07]. Meanwhile, the variance measures the distance of the trained model responses to their average value on the chosen data set. It shows how much the trained model (weights) can be sensitive to the data set choice [Bis07].