The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks

Phuong Ha Nguyen[1]*, Durga Prasad Sahoo[2], Chenglu Jin[1], Kaleel Mahmood[1], Ulrich Rührmair[3] and Marten van Dijk[1]

[1] University of Connecticut,
{phuong_ha.nguyen,chenglu.jin,kaleel.mahmood,marten.van_dijk}@uconn.edu
[2] Bosch India (RBEI/ESY), dpsahoo.cs@gmail.com
[3] LMU München, ruehrmair@ilo.de

Abstract. The design of a silicon Strong Physical Unclonable Function (PUF) that is lightweight and stable, and which possesses a rigorous security argument, has been a fundamental problem in PUF research since its very beginnings in 2002. Various effective PUF modeling attacks, for example at CCS 2010 and CHES 2015, have shown that currently, no existing silicon PUF design can meet these requirements. In this paper, we introduce the novel Interpose PUF (iPUF) design, and rigorously prove its security against all known machine learning (ML) attacks, including any currently known reliability-based strategies that exploit the stability of single CRPs (we are the first to provide a detailed analysis of when the reliability based CMA-ES attack is successful and when it is not applicable). Furthermore, we provide simulations and confirm these in experiments with FPGA implementations of the iPUF, demonstrating its practicality. Our new iPUF architecture so solves the currently open problem of constructing practical, silicon Strong PUFs that are secure against state-of-the-art ML attacks.

Keywords: Arbiter Physical Unclonable Function (APUF), Majority Voting, Modeling Attack, Strict Avalanche Criterion, Reliability based Modeling, XOR APUF, CMA-ES, Logistic Regression, Deep Neural Network.

1 Introduction

A Physical Unclonable Function (PUF) is a fingerprint of a chip which behaves as a *one-way function* in the following way: it leverages process manufacturing variation to generate a unique function taking "challenges" as input and generating "responses" as output. Silicon PUFs were introduced in 2002 by Gassend et al. [GCvDD02] and are an emerging hardware security primitive in various applications such as device identification, authentication and cryptographic key generation [MV10, KGM+08, YMSD11, BFSK11]. The fundamental open problem of *strong* silicon PUFs is: How to design a PUF that is lightweight, *strong* and secure. Below we explain each of these three properties and discuss current PUF designs that do not satisfy all of these properties. We also propose a new PUF design, the Interpose PUF (iPUF) to solve this open problem.

Lightweight and Strong Design. A lightweight design must have a small number of gates (small area) and in addition only *limited digital computation* is allowed with only a small number of gate computations and without the need for accessing additional memory.

*This project was supported in part by the AFOSR MURI under award number FA9550-14-1-0351. Ulrich Rührmair gratefully acknowledges funding by the PICOLA project of the German Bundesministerium für Bildung und Forschung (BMBF).

This lightweight implementation gives a small hardware footprint and high throughput. A strong PUF is one that has a large number of Challenge-Response Pairs (CRPs) which are impractical to enumerate.[1]

Security Argument. We consider an adversarial model in which an attacker attempts to obtain a software model of a PUF by using information extracted from measured CRPs. Intrinsically, a PUF hides a "random" function and learning such functions from input-output pairs is the field of machine learning – therefore, security must be argued with respect to the best-known applicable machine learning techniques. In practice we measure the security of a PUF by the number of required CRPs for modeling the PUF with a sufficiently high prediction accuracy (e.g. 90%).

Current PUF Designs. Current strong PUFs designs can generally be placed into one of three categories. 1. Strong PUFs without rigorous security arguments, e.g., the Power Grid PUF [HAP09], Clock PUF [YKL+13], Crossbar PUF [RJB+11]. 2. Strong PUFs based on the Arbiter PUF (APUF) or APUF design which have been broken by machine learning based attacks as discussed in Section 2, e.g., the XOR APUF [SD07], the Feed-Forward PUF [LLG+05], LSPUF (Lightweight Secure PUF) [MKP08], Bistable Ring PUF [CCL+11], and MPUF (Multiplexer PUF) [SMCN18]. 3. Strong PUFs with security proofs but are not lightweight, e.g., LPN PUF (Learning Parity with Noise based PUF) [HRvD+17, JHR+17].

Machine Learning Categorization. In this paper we divide the best-known machine learning attacks on PUFs into two types. The first type, attacks which use non-repeated measurements of CRPs, are denoted as Classical Machine Learning (CML). We further describe various CML attacks in Section 2. The CML security arguments for the iPUF are based on the theoretical arguments developed for the x-XOR APUF. It has already been proven that for the x-XOR APUF, if x is large enough then it is secure against known CML attacks [Söl09, RSS+10, TB15]. In this paper, we prove an equivalence relationship between the (x, y)-iPUF and x-XOR APUF such that the developed theoretical framework for CML for the x-XOR APUF can be applied to the iPUF. Specifically, if y is chosen large then the iPUF is theoretically secure against CML, just like the x-XOR APUF. In this way, we can re-use the XOR APUF CML security framework without having to create and prove an entirely new CML security framework. However, regardless of how large x is, the x-XOR APUF is still vulnerable to a second type of machine learning attacks.

The second type of machine learning attacks we discuss in this paper are attacks that use repeated CRP measurements [DV13, Bec15] (see Section 2 for more details). We denote these types of attacks as reliability-based machine learning. In this paper we prove these attacks are unable to model the iPUF. Overall, the iPUF is secure against CML attacks and reliability-based machine learning attacks. Moreover, it is as lightweight as an XOR APUF. Hence, the iPUF replaces the XOR APUF as a next-generation secure lightweight strong PUF.

Contributions. In this paper, our contributions are as follows.

- *Reliability-based CMA-ES (Covariance Matrix Adaptation Evolution Strategy) attacks.* Through theoretical study, rigorous simulation and experimentation we explain why the most prevalent reliability-based machine learning attack works on APUFs and XOR APUF. Moreover, we show under what circumstances the attack fails. To the best of our knowledge, we are the first ones to work on these two problems. In addition, we propose enhanced reliability-based CMA-ES attacks which are much more powerful than the original one.

- *Interpose PUF (iPUF).* We propose a novel APUF based PUF design called the

[1] Assuming no additional physical limitation on the read-out speed of CRPs from the PUF.

iPUF. The iPUF is a lightweight, strong secure PUF design that solves the open fundamental problem of silicon PUFs. We show that the iPUF can prevent CML attacks that use derivative based calculations (i.e. Logistic Regression) as well as the main reliability-based machine learning attack (i.e. reliability-based CMA-ES). Moreover, by showing the similarity between the iPUF and XOR APUF, we can prove that the iPUF is secure against deep neural network attacks and CRP-based CMA-ES attacks which are the most powerful classical attacks applicable to the iPUF.

- *Open Source Library.* We provide open source code on GitHub[2] for all our simulations and experiments. This includes code for machine learning attacks on APUF, XOR APUF and iPUF in Matlab and `C#`. Code to create various PUF models on FPGA hardware is also given.

Paper Organization. The paper is organized as follows.

The background on machine learning attacks is presented in Section 2. Section 3 briefly introduces the APUF and iPUF designs. Here, we also explain the road map to prove the resistance of the iPUF against all known modeling attacks, and introduce the specific chosen set of parameters for a practical secure iPUF (see Section 3.4). After that, we give a short description of reliability-based CMA-ES on XOR APUF and introduce its enhanced variant on APUF and XOR APUF in Section 4.

Section 5 provides a comprehensive study of reliability-based CMA-ES on XOR APUF, i.e., why it works and when it fails. We move to the security analysis of iPUF regarding classical machine learning attacks and reliability-based machine learning attacks in Section 6. We explicitly provide the security conjecture used for iPUF security and the design philosophy of iPUF (see Section 6.1). We introduce the notions *Contribution* and *Equivalence* to show the relationship between iPUF and XOR APUF in Section 6.3. With the developed understanding of reliability-based CMA-ES, the knowledge of the equivalence between iPUF and XOR APUF, we now can show the resistance of the iPUF against all known attacks, i.e., the reliability-based CMA-ES attack, the special class of machine learning attacks developed only for iPUF (coined linear approximation attacks, see Section 4.3), the Logistic Regression (LR) attacks, the Deep Neural Network attacks, the CRP-based CMA-ES attacks, the Boolean function based attacks, Deterministic Finite Automata attacks and Perceptron attacks used for PAC learning framework in Section 6.5. Section 7 studies the strict avalanche criteria (SAC) and the reliability property of the iPUF.

We provide detailed simulations to support the analysis of iPUF in Section 8. Section 9 presents the results for the iPUF implemented in FPGA. Here, we experimentally and theoretically explain the impact of *bad* implementations and *good* implementations on the security of the iPUF. We conclude our paper in Section 10. Moreover, we also provide a brief description of APUF and XOR APUF in Appendix A.1, the influence of a challenge bit on the output of an APUF in Appendix A.2, Multiplexer PUF's vulnerability with respect to the reliability-based CMA-ES attack in Appendix A.3, and $\tilde{k}$-junta test in Appendix A.4.

2 Background on Machine Learning Attacks on PUFs

In this section we discuss background information related to the evolution of machine learning attacks on PUFs.

[2]URL: https://github.com/scluconn/DA_PUF_Library

2.1 Classical Machine Learning Attacks Background

Classical Machine Learning (CML) was first introduced in 2004, where the 64 bit APUF was shown to be vulnerable with respect to Support Vector Machine (SVM) [Lim04, LLG+05]. The most efficient CML attack on APUF and XOR APUF [RSS+10, TB15] is Logistic Regression (LR) which uses non-repeated measurements of CRPs. LR was used to break APUFs and XOR APUFs in 2010 [RSS+10]. Rührmair et al. [RSS+10] demonstrated how the x-XOR APUF for $x \leq 5$ can be modeled with at least 98% accuracy using LR with non-repeated measurements of CRPs. The state-of-the-art implementation of LR breaks the 64 bit x-XOR APUF up to $x \leq 9$ and a 128 bit x-XOR APUF up to $x \leq 7$ [TB15]. For larger x the XOR APUF has been shown to resist LR due to the subexponential relationship between x and the amount of training data required to model the XOR APUF [Söl09, RSS+10, TB15].

Many PUF designs based on the APUF can also be broken using CML. The Bistable Ring PUF borrows from the APUF design and can be broken like the APUF using a neural network [SH14] or SVM [XRHB15]. The Feed-Forward APUF (FFA) [LLG+05] is another variant of the APUF and can be broken using SVM, LR and ES (Evolution Strategy) for ≤ 8 feedforward positions [RSS+10]. For a larger number of feedforward positions the FFA becomes unreliable and is therefore not considered to be practical. The Lightweight Secure PUF (LSPUF) [MKP08] is a variant of the XOR APUF with multiple outputs. The LSPUF is based on the x-XOR PUF and consequently is vulnerable to LR [SNMC15] when $x \leq 9$.

Another type of classical machine learning is the Probably Approximately Correct (PAC) [GTS15, GTS16, GTFS16] attack. PAC learning provides a framework to learn the maximum number of CRPs required for modeling an XOR PUF, with given levels of accuracy and final model delivery confidence. The Perceptron algorithm used in the PAC learning framework is shown to be successful against the x-XOR APUF for $x \leq 4$ [GTS15] and the Deterministic Finite Automata (DFA) [GTS16] algorithm can be used to model APUF. We note that DFA in [GTS16] can be modified to further attack x-XOR APUF, but the necessary modification is not fully explained. If we compare the LR attack on XOR APUF in [Söl09, RSS+10, TB15] with the Perceptron attack or with DFA attack on XOR APUF in [GTS15], we can see that the LR attack is much more powerful. This is because that LR has a much smaller search space, uses a mathematical model of the XOR APUF, and uses gradient information in an optimization process. Therefore, we only focus on the LR attack for XOR APUF in this paper. In addition, there is one Boolean function based attack related to the PAC learning framework [GTFS16]. However we show that it is inapplicable to APUFs, XOR APUFs and iPUFs in Section 6.5.

In this paper we focus on the main classical machine learning attacks: LR, Deep Neural Network [GBC16] (DNN) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Han16]. LR is considered as the most powerful attack on XOR APUF because it is a gradient-based optimization and it uses the mathematical model of XOR APUF [Söl09, RSS+10]. Since DNN is considered as one of the most powerful black box attack (i.e., does not require any mathematical model of the PUF), it is an interesting case for study. CMA-ES is a gradient free algorithm (i.e., the gradient is not used for finding the desired model) and this property makes it different from algorithms that require gradient based computations (i.e., the LR algorithm). By considering LR, DNN and CMA-ES algorithms, we complete the study of classical machine learning attacks on APUF-based PUFs (i.e., XOR APUF and iPUF).

2.2 Reliability-based Machine Learning Attacks Background

The first reliability-based attack was proposed in [DV13], where the authors used reliability information and Least Mean Square optimization method to attack APUF only. In other

words, this attack in [DV13] is not applicable to XOR APUFs. Later on, Becker [Bec15] was able to break the XOR APUF (and as a consequence the LSPUF) with a linear complexity in x using a non-classical ML attack which uses the reliability information of CRPs and the CMA-ES method. Reliability information can be extracted from repeated measurements of responses belonging to the same challenge [DV13]. This allows an attacker to measure the sensitivity of a response to environmental noise caused by e.g. temperature and voltage variations.

In this paper we refer to attacks that use repeated CRP measurements as *reliability-based* machine learning attacks.[3]

Note that if responses are not sensitive to environmental noise, i.e., the PUF is very reliable, then Becker's reliability-based CMA-ES attack is not applicable. For this reason one may want to add digital circuitry to the PUF in the form of a fuzzy extractor [DRS04] or an interface that exploits the LPN problem [HRvD+17, JHR+17], but these techniques are not lightweight and therefore do not solve the stated fundamental problem. A more lightweight solution is presented in [WGM+17] where the reliability of an x-XOR APUF is enhanced using majority voting. However, this is not sufficiently lightweight as the same circuitry (including memory for storing a counter) needs to be executed repeatedly a large number of times which implies a large number of gate evaluations and reduction in throughput. But more importantly, majority voting does *not prevent* the reliability-based CMA-ES attack as the environment can be pushed to extremes in order to make the PUF more unreliable again. In [SMCN18], a new PUF design was introduced but it is also not secure against CMA-ES attack (see Algorithm 1 in Section 5 in [SMCN18] or Appendix A.3). Since we only focus on the PUF primitive, PUF-based protocols which can prevent the reliability-based CMA-ES attack are out of the scope of this paper. For example the Lockdown protocol in [YHD+16] is one of good candidates. However, the iPUF design we propose is more lightweight and simpler than Lockdown protocol because it does not require many hardware primitives such as a True Random Number Generator, counters etc (see [YHD+16]).

3 The Arbiter PUF and Interpose PUF

In this section, we briefly introduce the analytical delay model [Lim04] and the reliability model [DV13] of the APUF. We also discuss the basic design of the newly proposed (x, y)-iPUF. Descriptions of both designs are necessary to understand the effectiveness of ML attacks on PUFs in Section 4. The reader can find the detailed description of APUF and XOR APUF in Appendix A.1.

3.1 The APUF Linear Additive Delay Model

In [Lim04, LLG+05], an analytical model called the *Linear Additive Delay Model* was presented. As shown in [Lim04], the linear additive delay model of an APUF has the form:

$$\Delta = \mathbf{w}[0]\mathbf{\Psi}[0] + \cdots + \mathbf{w}[i]\mathbf{\Psi}[i] + \cdots + \mathbf{w}[n]\mathbf{\Psi}[n] = \langle \mathbf{w}, \mathbf{\Psi} \rangle, \tag{1}$$

where n is the number of challenge bits, $\mathbf{w}$ and $\mathbf{\Psi}$ are known as *weight and parity (or feature) vectors*, respectively. The parity vector $\mathbf{\Psi}$ is derived from the challenge $\mathbf{c}$ as

[3] Reliability information can be regarded as a type of 'side-channel' information which is extracted from CRPs alone without the use of extraneous equipment, whereas additional equipment is needed in power side-channel analysis etc. [RXS+14, TDF+14, TLG+15]. In this paper security is argued in an adversarial model where the attacker only has access to CRPs.

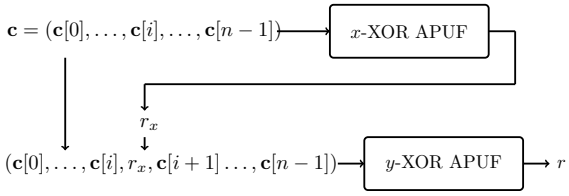


Figure 1: The (x, y)-iPUF (Interpose PUF) is comprised of an x-XOR APUF whose output r_x is interposed between $c[i]$ and $c[i+1]$ in the input of a y-XOR APUF.

follows:

$$\mathbf{\Psi}[n] = 1, \quad \text{and} \quad \mathbf{\Psi}[i] = \prod_{j=i}^{n-1}(1 - 2\mathbf{c}[j]), i = 0, \ldots, n-1. \tag{2}$$

In this delay model, the unknown weight vector $\mathbf{w}$ depends on the process variation of the APUF instance (i.e. of a specifically manufactured APUF). The response to a challenge $\mathbf{c}$ is defined as: $r = 0$ if $\Delta \geq 0$. Otherwise, $r = 1$.

3.2 The Arbiter PUF Reliability Model

Due to noise, the reproducibility or reliability of the output of the PUF is not perfect, i.e., applying the same challenge to a PUF will not produce a response bit with the same value every time. The repeatability is the *short-term reliability of a PUF* in presence of CMOS noise, and it is not the long-term device aging effect [DV13].

In an APUF we can measure the (short-term) reliability R (i.e., repeatability) for a specific challenge $\mathbf{c}$ in the following way: Assume that the challenge $\mathbf{c}$ is evaluated M times, and suppose the measured responses that are equal to 1 is N out of M evaluations. The reliability is defined as $R = N/M \in [0, 1]$.

In Eq. (1) we showed the mathematical relationship between the parity vector $\mathbf{\Psi}$ and the corresponding response Δ. Similarly, there exists a mathematical relationship between the reliability R of a given challenge and its response Δ as shown in [DV13]:

$$\Delta/\sigma_N = \sum_{i=0}^{n}(\mathbf{w}[i]/\sigma_N)\mathbf{\Psi}[i] = -\Phi^{-1}(R) \tag{3}$$

where the noise follows a normal distribution $\mathcal{N}(0, \sigma_N)$ and $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution. In the case where $R \in [0.1, 0.9]$ a further approximation [DV13] can be made:

$$\Delta/\sigma_N \approx R. \tag{4}$$

3.3 The iPUF Design

A (x, y)-iPUF consists of two layers. The upper layer is an n-bit x-XOR APUF and the lower layer is an $(n+1)$-bit y-XOR APUF. We denote the input to the x-XOR APUF as $\mathbf{c}_x = (\mathbf{c}[0], \ldots, \mathbf{c}[i], \mathbf{c}[i+1], \ldots, \mathbf{c}[n-1])$. The response r_x of the x-XOR APUF is interposed in $\mathbf{c}_x$ to create a new $n+1$ bit challenge $\mathbf{c}_y = (\mathbf{c}[0], \ldots, \mathbf{c}[i], r_x, \mathbf{c}[i+1], \ldots, \mathbf{c}[n-1])$. The final response bit is the response r_y of the y-XOR APUF with respect to the new challenge $\mathbf{c}_y$. The structure of the (x, y)-iPUF is shown in Fig. 1.

When creating an n-bit (x, y)-iPUF four important parameters determine its security against ML attacks. The four parameters are n which represents the length of the iPUF's challenge, x which represents the number of APUFs in the upper layer XOR APUF, y

which represents the number of APUFs in the lower layer XOR APUF, and the place where the response of the upper layer r_x is interposed in the input challenge for the lower layer y-XOR APUF. In the next two sections we will go over more details related to the ML attacks on PUFs including the (x, y)-iPUF. Based on this knowledge, we explain how to properly choose the design parameters for the (x, y)-iPUF in Section 6.5 to secure it against various ML attacks.

3.4 iPUF Security Philosophy

The iPUF design parameters are chosen based on certain machine learning attacks. In this section, we explain why we use certain specific attacks to dictate the choice of iPUF parameters instead of others. In Section 6 we will prove the (x, y)-iPUF is secure against reliability based CMA-ES attacks, so we only focus our design choices here based on classical machine learning attacks. Our explanation of design choices also requires us to develop a hierarchy to rank some of the classical machine learning attacks. In this manner, we can show that the iPUF can defend against the strongest classical attack.

Theorem 1. *For any given x-XOR APUF, the Logistic Regression (LR) attack is the most powerful attack among three classical machine learning attacks: the LR attack, the CRP-based CMA-ES attack and Deep Neural Network (DNN) attack.*

Proof. We note that the following comparison is only valid for XOR APUFs, and later we will link these comparison results to the security of iPUFs. The comparison is based on the following facts.

1. Fact 1. We can accurately describe the behavior of an XOR APUF using a mathematical model [Söl09, RSS+10].

2. Fact 2. The formulation of the LR attack exploits a-priori knowledge of the XOR structure in the XOR APUF, i.e., the mathematical model of the XOR APUF. The model used in the LR attack perfectly captures the structure of the XOR APUF [Söl09, RSS+10]. Moreover, LR is a gradient-based optimization methodology, i.e., for finding the true model, the gradient is based on a training dataset and the model is updated in each iteration. The gradient information will give the most *optimal updated direction* to guide the current model to the true model in each iteration. The reader is referred to [Sch16] for more details.

3. Fact 3. The CRP-based CMA-ES attack is a gradient free optimization methodology [Han16] and it uses the mathematical model of XOR APUF. In each iteration, it randomly generates a population of models based on the current model and then keeps only some best matching models among these models (see [Han16] or Section 4.1 for more details). The current model will be updated based on the information from these chosen models. Therefore, CRP-based CMA-ES only gives a *heuristic updated direction* to take the current model to the true model in each iteration. Therefore, CRP-based CMA-ES is not as efficient as LR (i.e., LR has the most optimal updated direction in each iteration as described in Fact 2). We also perform this attack on iPUF and XOR APUF in Section 6.3.

4. Fact 4. The deep neural network algorithm is a black box optimization methodology [GBC16], i.e., it does not use the mathematical model of XOR APUF. This is the main disadvantage compared to LR. Note that LR attack on XOR APUF takes the advantage of the mathematical model of XOR APUF (see Fact 2). Therefore LR does not have to optimize as many parameters as a DNN to model a PUF. In other words, the LR attack is more efficient than the DNN attack.

We formalize the result in the following inequalities of attacking efficiency.

$$\text{CMA-ES} < \text{Logistic Regression}. \tag{5}$$

$$\text{Deep Neural Network} < \text{Logistic Regression}. \tag{6}$$

□

Our experimental results in Table 3 also confirm our arguments above. Hence, the most powerful classical attack on XOR APUF is LR. However the LR attack is not applicable on iPUF as we will show in Section 6.5. Therefore we have to show the resiliency of the iPUF against the DNN attack and the CRP-based CMA-ES attack. To the best of our knowledge, there is **no** theory which can give a **lower bound** on the required number of CRPs for training a machine learning model to learn an XOR APUF with a given prediction accuracy using CRP-based CMA-ES, the DNN attack, or LR. Since the structure of iPUF is only slightly different from the structure of XOR APUF, we do not have such theory for iPUF either.

The x-XOR PUF has one theory which states that the number of CRPs required for the LR algorithm exponentially increases when x increases [Söl09]. Moreover, this theory was confirmed by experiments in [TB15] and the *empirical* lower bounds for the number of CPRs required for the LR algorithm is presented in [TB15].

Fortunately, we can come up with an "approximate" theory to choose the iPUF parameters based on the following information. First we know from the above attack comparison, LR is the strongest classical attack on an XOR APUF. We also know the XOR APUF's state-of-the-art empirical lower bound of the number CRPs to make the attack work [TB15]. Second we know the security of the iPUF can be related to the security of XOR APUF (which we formally develop in Theorem 4). The result in brief is that the (x, y)-iPUF with $i = n/2$ is equivalent to $(y + \frac{x}{2})$-XOR APUF (see Theorem 4).

Based on the two pieces of information above, we can conclude that if a $(y + \frac{x}{2})$-XOR APUF can be secure against LR, then the (x, y)-iPUF with $i = n/2$ can be secure against DNN attacks and CRP-based CMA-ES attacks as well. The literature has shown that a 64-bit 10-XOR APUF is secure against LR attack [TB15]. Based on that design **we suggest using challenge length $n = 64$, (1,10)-iPUF with interpose bit in the middle $i = n/2 = 32$ as a set of practical design parameters.**

4 Reliability Based ML Attacks

Conceptually, instead of using CRPs like in Classical ML (CML) attacks, the *repeatability* (i.e. short-term reliability) of APUF outputs can be used to build an APUF model based on a set of challenge-reliability (not response) pairs. The relationship between reliability, R and the weights $\mathbf{w}$ of an APUF were shown in Eq. (3) and Eq. (4). In [DV13] a Reliability based ML (RML) attack was done under the assumption that $R \in [0.1, 0.9]$. Based on this assumption a system of linear equations was established using Eq. (4) so that $\mathbf{w}[i]/\sigma_N$ could be solved for by using the *Least Mean Square* algorithm. This approach only applies to the APUF, and not the XOR APUF and by extension it also does not apply to the iPUF. However, RML attacks can be done **without** assumptions about the range of R, as we will explain in Section 4.1. Section 4.2 presents the enhanced version of this attack on APUF and XOR APUF. In Section 4.3 we show how to model the iPUF as a linear approximation (LA) of an XOR APUF. This can be used to analyze to what extent the CML and RML attacks on the XOR APUF apply to the iPUF.

4.1 Reliability Based CMA-ES

In [Bec14, Bec15] an ML attack on APUFs was developed using CMA-ES with reliability information obtained from the repeated measurements of CRPs. More precisely, the reliability information R of a challenge $\mathbf{c}$ (i.e. $(\mathbf{c}, R)$) is used in the attack instead of the corresponding response r (i.e. $(\mathbf{c}, r)$).

The rationale behind this attack is as follows: if the delay difference $|\Delta|$ between the two competing paths in an APUF for a given challenge $\mathbf{c}$ is smaller than a threshold ϵ, then the corresponding response r would be unreliable in the presence of noise; otherwise the response would be reliable. This implies that reliability information directly leaks information about the "wire delays" in an APUF model. Let r_i be the i-th measured response of challenge $\mathbf{c}$ for $i = 1, \ldots, M$. Two different definitions of R, as provided in Eq. (7) and Eq. (8), are found in [DV13] and [Bec15], respectively:

$$R = \frac{1}{M}\sum_{i=1}^{M} r_i, \tag{7}$$

$$R = |M/2 - \sum_{i=1}^{M} r_i|. \tag{8}$$

Note similar to Eq. (7) and Eq. (8), repeatability for an APUF was defined as $R = N/M \in [0, 1]$ (see Section 3.2). The objective of CMA-ES is to learn weights $\mathbf{w} = (\mathbf{w}[0], \ldots, \mathbf{w}[n])$ together with a threshold value ϵ. All variables $\mathbf{w}[0], \ldots, \mathbf{w}[n]$ are treated as independent and identically distributed Gaussian random variables. The attack is conducted as follows:

1. Collect N challenge-reliability pairs

$$\mathcal{Q} = \{(\mathbf{c}_1, R_1), \ldots, (\mathbf{c}_i, R_i), \ldots, (\mathbf{c}_N, R_N)\},$$

   where R_i is computed as in Eq. (8).

2. Generate K random models:

$$\{(\mathbf{w}_1, \epsilon_1), \ldots, (\mathbf{w}_j, \epsilon_j), \ldots, (\mathbf{w}_K, \epsilon_K)\}.$$

3. For each model $(\mathbf{w}_j, \epsilon_j)$ $(j = 1, \ldots, K)$, do the following steps:

   (a) For each challenge $\mathbf{c}_i$ $(i = 0, \ldots, N)$, compute the R'_i as follows:

$$R'_i = \begin{cases} 1, & \text{if } |\Delta| \geq \epsilon \\ 0, & \text{if } |\Delta| < \epsilon, \end{cases} \tag{9}$$

   where $\epsilon = \epsilon_j$ and Δ follows from (1) and (2) with $\mathbf{c} = \mathbf{c}_i$ and $\mathbf{w} = \mathbf{w}_j$. Note that for a given model $\mathbf{w}_j$ with input $\mathbf{c}_i$, R'_i indicates whether the output of the response of the model is reliable or noisy.

   (b) Compute the Pearson correlation coefficient ρ_j based on $(R_1, \ldots, R_i, \ldots, R_N)$ and $(R'_1, \ldots, R'_i, \ldots, R'_N)$,

   where R_i is computed as in Eq.(8).

4. CMA-ES keeps L models $(\mathbf{w}_j, \epsilon_j)$ which have the highest Pearson correlation ρ, and then, from these L models, another K models are generated based on the CMA-ES algorithm.

5. Repeat steps (3)-(4) for T iterations and model $(\mathbf{w}, \epsilon)$ which has highest Pearson correlation ρ will be chosen as the desired model. Note that sometimes the chosen model may have low prediction accuracy and in this case we restart the algorithm to find a model with higher prediction accuracy.

While the above pseudo-code is used to model an APUF, it can also be used to model an x-XOR APUF. Let

$$\mathcal{Q} = \{(\mathbf{c}_1, R_1), \ldots, (\mathbf{c}_i, R_i), \ldots, (\mathbf{c}_N, R_N)\}$$

be a set of challenge-reliability pairs of an x-XOR APUF instance. If the CMA-ES algorithm for modeling an APUF is executed many times with the set $\mathcal{Q}$, then it can produce x different models for $A_0, \ldots, A_{x-1}$ with high probability [Bec15]. Although there is no proof on how many times the CMA-ES algorithm has to be executed to get the models of all APUF instances of the x-XOR APUF, experimentally it is observed that this value needs not to be large, i.e., $2x$ or $3x$ CMA-ES runs. As done in [Bec15], one can parallelize CMA-ES executions to build models for $A_0, \ldots, A_{x-1}$.

As explained above, the modeling of an x-XOR APUF simplifies to the modeling of x independent APUF instances in the reliability based CMA-ES attack. This is significant because the relationship between x and the number of CRPs needed for the reliability based CMA-ES attack is linear. As previously stated, CML attacks have an exponential relationship between the number of CRPs and x. Therefore increasing x is a valid way to mitigate the CML attack. However, the reliability based CMA-ES attack cannot be defeated in the same manner.

4.2 Enhanced Reliability Based CMA-ES Attack On APUF and XOR APUF

In the reliability based CMA-ES attack, each model $\mathbf{w}$ is evaluated according to a fitness function. The fitness function correlates the observed reliability R of the PUF to the reliability of the estimated model R'. Let us denote the observed reliability R in Eq. (8) by R_{original} and the model reliability R' in Eq. (9) as R'_{original}. The correlation between R_{original} and R'_{original} is limited by the fact that the observed reliability R_{original} only ranges from $[0, M/2]$, while the range of R'_{original} is $\{0, 1\}$. This is significant because the comparison of R'_{original} and R_{original} is not as accurate, since the two terms do not have the same range of values.

We propose two alternative definitions for $(R_{\mathsf{original}}, R'_{\mathsf{original}})$. We define $(R_{\mathsf{absolute}}, R'_{\mathsf{absolute}})$ and $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$, as follows:

$$R_{\mathsf{original}} = |M/2 - \sum_{i=1}^{M} r_i| \qquad \text{and} \qquad R'_{\mathsf{original}} = \begin{cases} 1, & \text{if } |\Delta| \geq \epsilon \\ 0, & \text{if } |\Delta| < \epsilon, \end{cases} \tag{10}$$

$$R_{\mathsf{absolute}} = |M/2 - \sum_{i=1}^{M} r_i| \qquad \text{and} \qquad R'_{\mathsf{absolute}} = |\Delta| \tag{11}$$

$$R_{\mathsf{cdf}} = \frac{1}{M} \sum_{i=1}^{M} r_i \qquad \text{and} \qquad R'_{\mathsf{cdf}} = \Phi(-\Delta/\sigma_N) \tag{12}$$

We hypothesize that $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$ can create a more accurate model than $(R_{\mathsf{original}}, R'_{\mathsf{original}})$ when attacking an individual APUF for two reasons. First $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$ uses the proper reliability ranges. Second $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$ takes into account information from two sources: reliability information and the response information of the APUF by not using any absolute value operation when computing R'_{cdf}. However, when attacking an XOR APUF, the

response of each individual APUF is not known so $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$ does not improve the reliability based CMA-ES attack in this case.

When attacking an XOR APUF $(R_{\mathsf{absolute}}, R'_{\mathsf{absolute}})$ outperforms $(R_{\mathsf{original}}, R'_{\mathsf{original}})$ because it has the proper reliability ranges. Since it does not attempt to use any response information (which is not available from individual APUFs in an XOR APUF), it also outperforms $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$.

The simulated results of the enhanced attacks on APUF and XOR APUFs are presented in Section 8.3 and Section 8.4, respectively.

4.3 Linear Approximation of the (x, y)-iPUF

A technique that can be used in conjunction with CML or RML is linear approximation; this technique is specifically designed for use against the (x, y)-iPUF. We develop the technique based on the following observation: The difference between the input challenge to the y-XOR APUF in an (x, y)-iPUF and the input to a standard y-XOR APUF is only one bit. Assume $i + 1$ is the interposed bit position for the input challenge to the y-XOR APUF in an (x, y)-iPUF. The input to the y-XOR APUF is denoted as $\mathbf{c}_{low} = (\mathbf{c}[0], \ldots, \mathbf{c}[i], r_x, \mathbf{c}[i + 1], \ldots, \mathbf{c}[n - 1])$. Instead of attempting to learn the x-XOR APUF in the (x, y)-iPUF to estimate r_x, we can give a fixed value for bit $i + 1$, i.e. $\mathbf{c}'_{low} = (\mathbf{c}[0], \ldots, \mathbf{c}[i], 0, \mathbf{c}[i + 1], \ldots, \mathbf{c}[n - 1])$.

By making this approximation we can effectively ignore the x-XOR APUF component of the (x, y)-iPUF and treat the (x, y)-iPUF as a y-XOR APUF. Through this approximation we can do both CML and RML attacks on the (x, y)-iPUF while still using the XOR APUF model. These attacks are denoted as LA-CML and LA-RML. In Section 6.4.2 we analyze under what conditions the linear approximation accurately approximates the (x, y)-iPUF and how this attack can be mitigated.

5 Analysis of the Reliability based CMA-ES Attack

The reliability based CMA-ES attack is a serious security issue when designing a PUF. Our goal is to create a secure PUF design (the (x, y)-iPUF) that *prevents* this attack. To do this, it is necessary to understand under what conditions the reliability based CMA-ES attack works. In this section, we consider the following questions for an in-depth analysis of the reliability based CMA-ES attack:

1. In [Bec15] it is noted that CMA-ES converges more often to some APUF instances than others when modeling the components of an x-XOR APUF. Essentially this means some APUFs in an x-XOR APUF are *easier* and some are *harder* to model using the given challenge-reliability pairs. Why does this happen?

2. What are the conditions such that CMA-ES never converges to a particular APUF instance? In other words, under which condition does the reliability based CMA-ES attack fail?

The first question was posed in [Bec15] without any theoretical answer and the second question has not been investigated in literature. The next experiments are aimed at answering these questions. Based on the knowledge gained from these experiments, we develop the (x, y)-iPUF that is secure against the reliability based CMA-ES attack as well as the other known classical machine learning attacks mentioned in Section 2.

5.1 Experiment-I: Understanding CMA-ES Convergence

The objective of this experiment is to analyze why some APUF instances are *easier* and some are *harder* to model using reliability based CMA-ES. More specifically, we want to

Table 1: Relationship between the APUF model converged to by CMA-ES and the APUF's data set noise rate proportion in each data set $\mathcal{Q}$

Rank	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$	$n=10$	Failed
Occurrence	36%	5%	9%	5%	12%	7%	7%	5%	7%	5%	2%

Rank: APUF model had the n-th highest data set noise proportion in the data set.
Failed: Failed to converge to any APUF model.
Occurrence: Percentage of time a convergence occurred.

verify whether the probability of converging to an APUF model in CMA-ES is correlated with the "data set noise proportion" of that APUF instance present in a given data set $\mathcal{Q}$. Here we define the data set noise proportion for a particular APUF in an x-XOR APUF as the number of noisy (unreliable) CRPs $\mathcal{Q}_n$ due to the specified APUF divided by the number of total CRPs in $\mathcal{Q}$. We define the *noise rate* β of a given PUF as follows. Let $r_{\mathbf{c},1}$ and $r_{\mathbf{c},2}$ be the values of the first and the second measurements of the PUF for a given challenge $\mathbf{c}$, respectively. The noise rate β of the PUF is equal to $Pr_{\mathbf{c}}(r_{\mathbf{c},1} \neq r_{\mathbf{c},2})$, i.e., $\beta = Pr_{\mathbf{c}}(r_{\mathbf{c},1} \neq r_{\mathbf{c},2})$.

Every time we generate a new set $\mathcal{Q}$, the challenge-reliability pairs $\{(\mathbf{c}_i, R_i)\}$ of $\mathcal{Q}$ can be divided into two parts. The first part is made up of the reliable challenge-reliability pairs $\mathcal{Q}_r$ and the second part is made up of the noisy challenge-reliability pairs $\mathcal{Q}_n$. Each APUF instance A_i has its own set of noisy challenge-reliability pairs $\mathcal{Q}_{i,n} \subset \mathcal{Q}_n, i = 0, \ldots, x-1$.

Theorem 2. *For a given dataset $\mathcal{Q}$, the CMA-ES algorithm is more likely to converge to the APUF instance which has the largest number of pairs in the combined set $\mathcal{Q}_r \cup \mathcal{Q}_{i,n}$, i.e., highest Pearson correlation, where $\mathcal{Q}_r$ is the reliable CRPs and $\mathcal{Q}_{i,n}$ is the noisy CRPs for APUF instance i.*

Proof. Since $\mathcal{Q}_r$ is useful for modeling all the APUF instances, the set $\mathcal{Q}_{i,n}$ should be large to make the $\mathcal{Q}_r \cup \mathcal{Q}_{i,n}$ sufficiently large enough for modeling the i-th APUF instance. From Step-4 of the reliability based CMA-ES attack in Section 4.1, it is obvious that CMA-ES tries to converge to the APUF instance that has the largest value for $|\mathcal{Q}_{i,n}|/|\mathcal{Q}|$. □

To verify Theorem 2, we do the following experiment.

Experimental Setup: We run the CMA-ES algorithm 100 times on a 10-XOR APUF in simulation. Each time we run CMA-ES we use a new randomly generated dataset $\mathcal{Q}$ which contains 70×10^3 CRPs. The noise rate of each individual APUF is 20%. *Note that the 20% noise rate is chosen to be intentionally large to make the number of used CRPs small, i.e., the bigger the noise of each APUF, the less number of CRPs required for the reliability-based attack. This is by no means the only possible noise rate that could work, but we only use this noise rate here to illustrate the results of the experiment.* It is important to note that while the noise rate of each APUF is the same, the data set noise proportion of each APUF in $\mathcal{Q}$ will change each time we generate a new $\mathcal{Q}$. For each $\mathcal{Q}$, we generate the challenge-reliability pairs for $\mathcal{Q}$ by measuring the response to each challenge 11 times. Note the repeated measurement 11 is chosen heuristically here based on two criteria. The repeated measurement must be large enough to capture the reliability information with acceptable precision, but not too large that the repeated measurement time significantly slows down the experiment run time.

Experimental Results: The results of experiment I are summarized in Table 1. In each run of CMA-ES, a model $\mathbf{w}$ is generated. We classify the model in the following way: if the model $\mathbf{w}$ matches one of the APUF models with probability ≥ 0.9 or ≤ 0.1 (complement model), then we accept it as a *correct* model for that particular APUF instance. If the generated model $\mathbf{w}$ does not match any of the APUF models we consider the CMA-ES algorithm to have failed to correctly converge. Note that $\mathbf{w}$ can match with at most one APUF instance. This is because if the model $\mathbf{w}$ corresponds to a particular APUF instance, then only that instance will have a matching probability ≥ 0.9 or ≤ 0.1, and the other APUF instances will have a matching probability around 0.5. This is due to the good

uniqueness property of simulated APUF instances. *Note that in Experiments I and II we are not a true adversary, i.e., we are a verifier regarding the reliability based CMA-ES attack on the x-XOR PUF as described in [Bec15]. In other words, we know all APUF instances and check whether a model produced by the CMA-ES algorithm is a correct model for one of them. The audience is referred to [Bec15] for the detailed description of a real attack.*

For the given $\mathcal{Q}$, we measure the data set noise proportion of each APUF with respect to the challenges present in $\mathcal{Q}$. If CMA-ES converges to the APUF model with the highest data set noise proportion in the current $\mathcal{Q}$, we increment the count in column one. If CMA-ES converges to the APUF model with the second highest data set noise proportion in the current $\mathcal{Q}$, we increment the count in column two and so on. If the CMA-ES algorithm generates a model that corresponds to none of the APUFs then we say it failed to converge to any model.

Analysis of Results: The APUF with the highest data set noise rate proportion in $\mathcal{Q}$ should have the highest Pearson correlation coefficient as explained in Theorem 2 and therefore be the global maximum. However CMA-ES does not guarantee convergence to the global maximum. When CMA-ES converges to a local maximum, it produces another one of the valid APUF models, or an invalid model. For this reason we can see in Table 1 that we can converge to a PUF model that does not have the highest noise proportion with small probability compared to the highest case, i.e.,

$$\frac{5}{100}, \frac{7}{100}, \frac{9}{100}, \frac{12}{100} \ll \frac{36}{100}.$$

Overall every time we generate $\mathcal{Q}$, the PUF with the highest noise proportion in $\mathcal{Q}$ will have a high probability of being found by CMA-ES (i.e., 36/100). Likewise, the PUFs with a smaller noise proportion in $\mathcal{Q}$ will have a smaller probability of being found.

5.2 Experiment-II: CMA-ES Reliability Conditions

The objective of this experiment is to understand under which condition the reliability based CMA-ES attack fails to build a model for a particular APUF in an x-XOR APUF. From experiment I we know that CMA-ES is most likely to converge to the APUF model which has the highest data set noise proportion in $\mathcal{Q}$. We want to show that if the noise rate of the APUF instances in the x-XOR APUF's output are *not equal* (i.e., drawn from different distributions), then some APUF models cannot be generated by CMA-ES.

Experimental Setup: We simulate a 2-XOR APUF consisting of two APUF instances, denoted as A_0 and A_1. The noise rate of both APUFs are set at 1%. We run CMA-ES on the 2-XOR APUF 100 times. In each run of CMA-ES we generate a $\mathcal{Q}$ of size 70×10^3, where in $\mathcal{Q}$ each challenge is evaluated 11 times to generate the reliability information.

In this experiment we hypothesize that CMA-ES fails to build a certain APUF model when that model always has a lower noise rate (and therefore a lower data set noise proportion in $\mathcal{Q}$). To do this we manipulate the reliability information presented in the final output, such that A_0 always has lower data set noise proportion. This is achieved by applying majority voting to the responses of A_0 before XOR-ing it with the response of A_1. In majority voting, we have experimented with $M = 5$ and $M = 10$ votes to observe the performance of CMA-ES. *We note that the noise rate in this experiment (i.e., 1%) is not related to the noise rate in Experiment-I (i.e., 20%). Here, we intentionally choose a small noise rate to demonstrate two things. First even if the noise rate of the APUFs are extremely small, the CMA-ES attack still works. Second even if there is a sufficiently small discrepancy in the noise rates between two APUFs, CMA-ES will converge to the APUF model which is the nosiest. The numbers* 5 *and* 10 *for majority voting and* 11 *for*

Table 2: Results of Experiment-II

$M^{\dagger}$	Number of times A_0 found	Number of times A_1 found
5	8	92
10	0 ‡	99 ‡

† No. of measurements used in the majority voting of A_0.

‡ The sum of the two counts is not equal to 100 because one attack failed.

reliability evaluation are chosen arbitrarily. While other values are possible here, we just use one set of values to illustrate the results of our experiment.

Theorem 3. *In a 2-XOR APUF, if one APUF instance is sufficiently more reliable than the other, then the CMA-ES fails to converge to the APUF which has lower noise rate.*

Proof. We will prove the Theorem above using the experimental setup we described in the beginning of Section 5.2. If the number of votes M is sufficiently large, then the output of A_0 is always more reliable than that of A_1 due to the existing majority voting at the output of A_0. This implies that $|\mathcal{Q}_{0,n}|/|\mathcal{Q}| \ll |\mathcal{Q}_{1,n}|/|\mathcal{Q}|$ for any given $\mathcal{Q}$ where $\mathcal{Q}_{i,n}$ is the set of noisy challenge-reliability pairs of $A_i, i = 0, 1$. For a given data set $\mathcal{Q}$, as stated in Theorem 2, CMA-ES tends to converge to the model with the highest data set noise proportion with high probability. Hence, CMA-ES always converges to A_1 when M is sufficiently large. □

Experimental Results: The experimental results are shown in Table 2. The first column corresponds to the number of times the output was measured on A_0 before majority voting was done. The second and the third columns refer to the number of times the correct model was found by CMA-ES for A_0 and A_1, respectively.

Analysis of Results: From Table 2, it is clear that if M is sufficiently large ($M \geq 10$), then the reliability based CMA-ES attack cannot build a model for A_0. This is because we decreased the noise rate of A_0 by applying majority voting. Since A_0 is less noisy, it will have a lower data set noise proportion in $\mathcal{Q}$. As we established in the first experiment, CMA-ES tends to converge to the model with the highest data set noise proportion with high probability. In Table 2 we can clearly see this happening when $M = 10$, as CMA-ES is never able to build a model for A_0 (the APUF with the lower data set noise proportion).

It is important to also note that just because the APUFs have different noise rates, it does not make the XOR APUF secure against the reliability based CMA-ES attack. In the setup described in this experiment, an attacker could simply learn a model for A_1. Once the model for A_1 has been learned the attacker could then use that model to remove most of the noisy challenge-reliability pairs corresponding to A_1 from $\mathcal{Q}$. Doing this would create a new dataset $\mathcal{Q}_{reduced}$ from $\mathcal{Q}$ in which A_0 would have the highest data set noise proportion. The attacker could then run CMA-ES on $\mathcal{Q}_{reduced}$ to get a model for A_0.

5.3 Inferences from the Experiments

Two important points can be understood from the experiments in this section. In order for the reliability based CMA-ES attack to be successful the following conditions must be met:

1. All APUF instances outputs must have the same influence on the final output of the PUF.

2. The noise rate of all APUF instances should be similar.

In the next section, we leverage the knowledge from these experiments to show how the proposed iPUF can be secured against the reliability based CMA-ES attack.

6 Security Analysis of iPUF against Machine Learning Attacks

In this section, we analyze the security and reliability of the proposed (x, y)-iPUF design. Starting in Section 6.1 we discuss our basic conjecture to reason about the security of the iPUF design. In Section 6.2 we begin our security analysis by first showing how a single challenge bit in an APUF effects its output r. We then develop a relationship between the security of the (x, y)-iPUF and x-XOR APUF in Section 6.3. We then use the analysis from Section 6.2 to determine the most secure position for the interposed bit in a $(1, 1)$-iPUF in Section 6.4.2. Since the iPUF is developed to *prevent* the reliability-based ML attacks, we need to study the reliability of the $(1, 1)$-iPUF as shown in Section 6.4.1. After that, we explain why iPUF's design can *prevent* reliability-based ML attacks. While the $(1, 1)$-iPUF can prevent reliability-based ML attacks, properly choosing the interposed bit position alone is not enough. The $(1, 1)$-iPUF still suffers from classical machine learning, LA-CML and LA-RML attacks (see Section 4.3). Therefore, in Section 6.5 we expand our analysis to the (x, y)-iPUF where $x \geq 1$ and $y > 1$. By using a middle interposed bit and properly choosing x and $y \geq 2$ we are able to show that the (x, y)-iPUF is secure against all current state-of-the-art ML attacks. Based on our analysis we claim properly designed iPUFs are superior to XOR APUFs in terms of security, reliability and hardware overhead.

6.1 Security Conjecture and Design Philosophy of iPUF

Security Conjecture. Our basic security conjecture is *"if x is sufficiently large, then an x-XOR APUF is practically secure against all known classical ML attacks"* as stated in the literature [Söl09, RSS+10, TB15] (this is not an NP-hard problem). We show that the (x, y)-iPUF is equivalent to the $(x/2 + y)$-XOR APUF, (see Definition 2 for the definition of equivalence), so it inherits the same classical ML security. Furthermore the iPUF can prevent the reliability-based CMA-ES attack which the XOR APUF is vulnerable to. For PUF research, the XOR APUF w.r.t. classical ML is similar to AES (Advanced Encryption Standard) block cipher in the sense that it is considered a hard problem by itself [Söl09, RSS+10, TB15], i.e., the security of AES is not reduced to any intractable problem [DR02]. Indeed, we can show that among all known classical ML attacks on XOR APUF, LR attack is the most powerful attack (see Theorem 1) because it is a gradient-based optimization method and it uses a mathematical model of the XOR APUF. In other words, it uses the information about the XOR APUF (the XOR APUF mathematical model) in the most efficient way. No non-repeated measurement ML attack can match LR. Moreover, the LR attack on XOR APUFs has a full theory developed on the relationship between the modeling accuracy and modeling complexity [Söl09]. This theory is strongly supported with empirical results [TB15]. In summary, we deeply understand the resistance of XOR APUF against all known machine learning attacks.

Design Philosophy. Our design philosophy is as follows: We propose that it is possible to construct a strong PUF whose security can be reduced to an intractable problem (classical ML on XOR APUF). This practice of basing PUF security on its reduction to an intractable problem is not unique to this paper. For example the LPN-based PUF [HRvD+17, JHR+17] uses this exact same formulation and reduces its security to a well-known computational hardness assumption: learning parity with noise (LPN). Note that the large matrix operations in LPN-based PUF has an extremely large hardware footprint making it not lightweight and hence not a solution to the fundamental PUF problem considered in this paper. To develop a lightweight and secure strong PUF, we adopt another approach. We use broken but lightweight and reliable PUF primitives (i.e., APUF) to develop a secure, lightweight and reliable PUF design (i.e., iPUF) which can be shown to be equal to some *hard* problem (classical ML on an XOR APUF). We consider this

approach as a practical and proper methodology to develop a lightweight and secure strong PUF. To re-iterate, this process is commonly accepted in the cryptography community to develop cryptographic primitives such as block ciphers, stream ciphers and hash functions. In all these cryptographic primitives the secure designs are built on *insecure* components such as s-box, non linear shift register, etc [DR02, MOV96]. Moreover, the mathematical model of the APUF is fully developed and the iPUF is a simple composite APUF design. Hence, a full mathematical model of the iPUF can be developed for *analysis purpose*. This point is significantly important, because we can do a clean security analysis as we will show in the rest of the paper. In summary, we use APUF as a building block of iPUF because it is a lightweight, reliable strong PUF with a clean mathematical model for analysis purpose. Our iPUF design is also significantly simple. Hence, it also offers a clean framework for conducting security analysis.

6.2 Influence of Challenge Bit $\mathbf{c}[j]$ on the Response r in the APUF

The influence of a challenge bit on an APUF's response depends on its position in the challenge $\mathbf{c}$ [MKP09, DGSV14, NSCM16]. From Eq. (1), it can be observed that $\mathbf{\Psi}[j+1], \ldots, \mathbf{\Psi}[n-1]$ does not depend on challenge bit $\mathbf{c}[j]$. For a given challenge $\mathbf{c}$, based on the linear delay model of the APUF, the delay difference Δ can be described as:

$$\Delta = (1 - 2\mathbf{c}[j]) \times \Delta_{Flipping} + \Delta_{Non-Flipping} \tag{13}$$

where $\Delta_{Flipping}$ is the term affected by the flipping of bit $\mathbf{c}[j]$ and is given by $\Delta_{Flipping} = \sum_{i=0}^{j} \mathbf{w}[i] \frac{\mathbf{\Psi}[i]}{(1-2\mathbf{c}[j])}$. Likewise, the term that is not dependent on the flipping of $\mathbf{c}[j]$ is denoted as $\Delta_{Non-Flipping} = \sum_{i=j+1}^{n} \mathbf{w}[i]\mathbf{\Psi}[i]$.

If $\mathbf{c}[j] = 0$ then $\Delta = \Delta_{Flipping} + \Delta_{Non-Flipping}$ and we will denote this Δ as $\Delta_{\mathbf{c}[j]=0}$ with corresponding response $r_{\mathbf{c}[j]=0}$. Similarly when $\mathbf{c}[j] = 1$ we will have $\Delta = -\Delta_{Flipping} + \Delta_{Non-Flipping}$. We denote this Δ as $\Delta_{\mathbf{c}[j]=1}$ and the response as $r_{\mathbf{c}[j]=1}$.

We want to know the influence of flipping $\mathbf{c}[j]$ on output r. We measure this influence by computing the probability that the output remains the same if we flip bit $\mathbf{c}[j]$ while keeping the rest of $\mathbf{c}$ constant:

$$\Pr_{\mathbf{c}}(r_{\mathbf{c}[j]=0} = r_{\mathbf{c}[j]=1}). \tag{14}$$

In Appendix A.2, we explain:

$$\Pr_{\mathbf{c}}(r_{\mathbf{c}[j]=0} = r_{\mathbf{c}[j]=1}) \approx \frac{(n-j)}{n}, j = 0, \ldots, n-1 \tag{15}$$

This implies that an expected probability $\Pr_{\mathbf{c}}(r_{\mathbf{c}[j]=0} = r_{\mathbf{c}[j]=1})$ decreases with increasing value of j, so the influence of each challenge bit is not equal. This undesirable security property means we must carefully consider the position of the interposed bit. In the next section we analyze how the interposed bit position effects the security and reliability of the $(1,1)$-iPUF. We note that the influence of individual challenge bits on the response is highly related to the SAC (Strict Avalanche Criterion) introduced in [MKP09, DGSV14, NSCM16], and the SAC property of iPUF will be further explained and analyzed in Section 7.2.

6.3 The Relationship between (x, y)-iPUF and $(x + y)$-XOR APUF

For a fixed challenge we study the contribution of each APUF component of an APUF-based PUF (i.e. XOR APUF or iPUF) to its output. Here we define **contribution** in the following manner.

Definition 1. **[Contribution]** For a given challenge $\mathbf{c}$, if the output of an APUF A_i flips (while the rest of the APUF outputs are held constant) and this causes the final response of the APUF-based PUF r to flip, then we say that A_i **contributes** to the output for the challenge $\mathbf{c}$.

In an x-XOR APUF it can clearly be seen that each of the x APUFs contribute to the output. This is because in an XOR gate flipping any one of the inputs always causes the output to flip. We know the more APUFs that contribute to the output, the harder the XOR APUF design is to attack with classical machine learning.

Definition 2. **[Equivalence]** For a given challenge $\mathbf{c}$, if an APUF-based PUF has m APUFs contributing to the final response, then we say that this PUF is equivalent to an m-XOR APUF for the challenge $\mathbf{c}$

Theorem 4. *If the interposed bit position is i, then averaged over all possible challenges $\mathbf{c}$ the n-bit (x, y)-iPUF is equivalent to the n-bit $(y + p_r x)$-XOR APUF where*

$$p_r = \frac{1-(1-2p)^y}{2} \qquad \textit{and} \qquad p = \frac{i}{n}.$$

Proof. We denote r_y^0 as the output of the iPUF when $r_x = 0$ with $\mathbf{c}_y = (\mathbf{c}[0], \ldots, \mathbf{c}[i], r_x = 0, \mathbf{c}[i+1], \ldots, \mathbf{c}[n-1])$ and r_y^1 as the output of the iPUF when $r_x = 1$ with $\mathbf{c}_y = (\mathbf{c}[0], \ldots, \mathbf{c}[i], r_x = 1, \mathbf{c}[i+1], \ldots, \mathbf{c}[n-1])$. Based on our definition of contribution above, if $r_y^0 = r_y^1$ it means that the x-XOR APUF output r_x does not contribute to the final output r_y of the iPUF. In this case only y APUFs contribute to the output of the iPUF. Note we can also write $r_y^0 = r_y^1$ as $r_y^1 \oplus r_y^0 = 0$. Alternatively if $r_y^1 \oplus r_y^0 = 1$ then the output of (x, y)-iPUF depends on the output r_x of x-XOR PUF, as well as the output r_y of the y-XOR PUF. This implies that there are $(x + y)$ APUFs which contribute to the final output r_y for a given challenge. Therefore, the challenge-response space of an (x, y)-iPUF can be partitioned into two groups. The first group represents challenge-response pairs where the response only depends on the y APUFs of the y-XOR PUF. The second group of challenge-response pairs has responses which depend on both the x-XOR APUF and the y-XOR APUF (the response depends on a total of $(x + y)$ APUFs). Now we calculate the expected number of challenge-response pairs in each group. First, we will compute the probability the challenge-response pair is in the second group:

$$p_r = \Pr_{\mathbf{c}}(r_y^0 \neq r_y^1) = \Pr_{\mathbf{c}}(r_y^0 \oplus r_y^1 = 1) \tag{16}$$

Let $r_{low,0}, r_{low,1}, \ldots, r_{low,y-1}$ be the outputs of the y APUFs in the lower layer y-XOR PUF when r_x is 0. We will assume ideal classical machine learning conditions where no measurement noise is present. Because there is no measurement noise Section 6.2 is applicable: For a given challenge $\mathbf{c}$, $r_{low,i}$ depends on the upper layer output r_x (in that output $r_{low,i}$ would flip if r_x would be substituted by 1) with probability $p = (i+1)/(n+1) \approx i/n$ if the feedback position of r_x is i. For a given challenge $\mathbf{c}$, p_r is the probability that the response of (x, y)-iPUF is equal to $r = 1 \oplus r_{low,0} \oplus \ldots \oplus r_{low,y-1}$ if r_x would be substituted by 1. Then p_r depends on i:

$$p_r = \sum_{k=1,k \text{ odd}}^{y} \binom{y}{k} p^k (1-p)^{y-k} = {}^{4}\frac{1-(1-2p)^y}{2}. \tag{17}$$

[4]For any given a and b, we have:

$$(a+b)^x - / + (a-b)^x = 2 \sum_{j=0,j \text{ is odd/ even}}^{x} \binom{x}{j} a^j b^{x-j}.$$

Hence, replacing $a = 1 - p$ and $b = p$ yields Eq.(17).

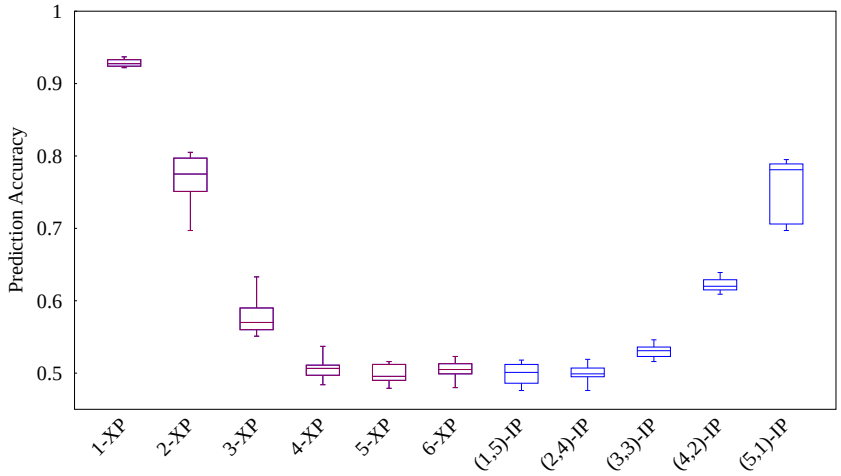


Figure 2: Prediction accuracy of CRPs based CMA-ES attack on 64-bit APUF (1-XP), 64-bit 2,3,4,5,6-XOR APUF (y-XP) and (1,5), (2,4), (3,3), (4,2), (5,1)-iPUF ((x,y)-IP).

Thus, a fraction $1 - p_r$ of challenge-response pairs is in the first group and a fraction p_r of challenge-response pairs is in the second group. For a given (x, y)-iPUF with parameter i, the *expected* number of APUFs contributing to the response of a given challenge is

$$\begin{aligned} &= (1 - p_r)y + p_r(x + y) \\ &= y + p_r x \end{aligned} \tag{18}$$

□

Simulation Validation. Through simulation we compare the (x, y)-iPUF and the $(x+y)$-XOR APUF. Since we only focus on the design, we consider reliable PUFs to experimentally demonstrate Eq. (18), i.e. If the interposed bit position is in the middle, then the (x, y)-iPUF is equivalent to a $(y + \frac{x}{2})$-XOR APUF (see Theorem 4). We run the CRPs-based CMA-ES attack on 64-bit APUF (a.k.a 1-XOR APUF or 1-XP), 64-bit 2,3,4,5,6-XOR APUF (y-XP) and (1,5), (2,4), (3,3), (4,2), (5,1)-iPUF ((x,y)-IP) with 200,000 CRPs for training. In each attack the CMA-ES algorithm is run for 1000 iterations. The results are shown in Fig 2. For each PUF, we attack it 10 times. The prediction accuracy of each attack is computed using 2000 CRPs. Fig 2 shows that the experimental results closely match the theory presented in Eq. (18). Note that the prediction accuracy of (5,1)-iPUF is higher than that of 3-XOR APUF or 4-XOR APUF, because when the lower layer only has one APUF, the linear approximation attack becomes effective. Here the attacker only needs to model the APUF in the lower layer accurately, after which the upper bound on the accuracy for the linear approximation attack on $(x, 1)$-iPUF can be achieved, which is 75% accuracy. A detailed analysis of the linear approximation attack will be provided in Section 6.4.2.

6.4 Security and Reliability Analysis of the (1,1)-iPUF

The most basic form of the (x, y)-iPUF is the $(1, 1)$-iPUF, with the upper layer consisting of a single APUF A_{up} and the lower layer consists of a single APUF A_{low}. Let us denote the responses to A_{up} and A_{low} by r_{up} and r_{low}, respectively. The final output of the $(1, 1)$-iPUF is the response r_{low}. Based on the $(1, 1)$-iPUF structure we analyze where to interpose the bit in the lower APUF and how this affects the reliability and security of the $(1, 1)$-iPUF.

6.4.1 Reliability of the (1,1)-iPUF

In order to determine the effect of measurement noise in the $(1,1)$-iPUF, we evaluate a challenge $\mathbf{c}$ twice. Let us denote $r_{up,0}$ as the response of the upper APUF and $r_{low,0}$ as the response of the lower APUF the first time the challenge is applied. Likewise, let us denote $r_{up,1}$ and $r_{low,1}$ as the APUF's responses the second time the challenge is evaluated.

Assume APUF A_{up} has noise rate β_{up} such that $\Pr_{\mathbf{c}}(r_{up,0} \neq r_{up,1}) = \beta_{up}$. Similarly, assume that APUF A_{low} has noise rate β_{low} such that $\Pr_{\mathbf{c}}(r_{low,0} \neq r_{low,1} | r_{up,0} = r_{up,1}) = \beta_{low}$.

Let us denote $i+1$ as the interposed bit position for r_{up} in the $(n+1)$-bit challenge of A_{low}. If the flipping of the output of A_{up} flips the output of A_{low} and A_{low} is itself reliable, then the output of A_{low} should be flipped. This event occurs with probability $(1-\beta_{low})\frac{i+1}{n+1}$. If the flipping of the output of A_{up} does not flip of the output of A_{low} and A_{low} is itself noisy, then the output of A_{low} flips. This event occurs with probability $\beta_{low}\left(1-\frac{i+1}{n+1}\right)$. Therefore, the event where flipping the output of A_{up} flips the output of A_{low} occurs with the following probability

$$\Pr_{\mathbf{c}}(r_{low,0} \neq r_{low,1} | r_{up,0} \neq r_{up,1}) = (1-\beta_{low})\frac{i+1}{n+1} + \beta_{low}\left(1-\frac{i+1}{n+1}\right).$$

We use the equality above and Eq. (15) to derive

$$\begin{aligned}
& \Pr_{\mathbf{c}}(r_{low,0} \neq r_{low,1}) \\
= \; & \Pr_{\mathbf{c}}(r_{low,0} \neq r_{low,1} | r_{up,0} = r_{up,1}) \Pr_{\mathbf{c}}(r_{up,0} = r_{up,1}) + \\
& \Pr_{\mathbf{c}}(r_{low,0} \neq r_{low,1} | r_{up,0} \neq r_{up,1}) \Pr_{\mathbf{c}}(r_{up,0} \neq r_{up,1}) \\
= \; & \beta_{low}(1-\beta_{up}) + [(1-\beta_{low})\frac{i+1}{n+1} + \beta_{low}\left(1-\frac{i+1}{n+1}\right)]\beta_{up} \\
= \; & \beta_{low}(1-\beta_{up}) + [(1-2\beta_{low})\frac{i+1}{n+1} + \beta_{low}]\beta_{up}. \qquad (19)
\end{aligned}$$

In practice $\beta_{up} \ll 1$ and $\beta_{low} \ll 1$, thus Eq. (19) can be approximated as:

$$\Pr_{\mathbf{c}}(r_{low,0} \neq r_{low,1}) \approx \beta_{low} + \beta_{up}\frac{i}{n} \qquad (20)$$

From Eq. (19) and Eq. (20) it can be seen that the reliability information of A_{up} and A_{low} available at the output of $(1,1)$-iPUF are not *equal* even when $\beta_{low} = \beta_{up}$. If we assume $\beta_{low} = \beta_{up} = \beta$ then A_{low} contributes approximately β while A_{up} contributes approximately $\beta\frac{i}{n}$. The unequal reliability contribution shown by our analysis has important implications for the success of the reliability based CMA-ES attack.

6.4.2 Security of $(1,1)$-iPUF

We will now discuss the security of the (1,1)-iPUF with respect to the reliability based CMA-ES attack (i.e. RML), CML and ML attacks that use the linear approximation technique (i.e. LA-CML and LA-RML).

Reliability Based CMA-ES Attack: The (1,1)-iPUF is theoretically secure against the CMA-ES reliability attack for two reasons if the interposed bit position for r_{up} is properly chosen: The first reason is based on the conditions under which the CMA-ES attack operates and the second is based on the computation done to learn the APUF model in CMA-ES.

First, recall the conditions under which the reliability based CMA-ES attack works as described in Section 5.2. In order to successfully model the APUF components A_{up}

and A_{low}, each APUF must contribute equal reliability information to the output. For the (1,1)-iPUF, we showed in Eq. (20) that the contributions of A_{low} and A_{up} are not equal, i.e., β is not equal to $\beta\frac{i}{n}$ when i is between 0 and $\frac{n}{2}$. Due to the unequal contribution of reliability information in the output when the interposed bit position $(i+1)$ is not close to n, CMA-ES will not converge to the model for A_{up}. We did the following experiment to determine the importance of the interposed bit position. We launched the reliability based CMA-ES attack on a 64-bit (1,1)-iPUF with 30,000 challenge-reliability pairs. In this experiment, the noise rate of each APUF was 20% ($\beta = 0.2$). The number of iterations for CMA-ES was 30,000. We repeated the attack 20 times with the interposed bit position at 0 ($i = 0$), i ($i = n/2$) and 64 ($i = n$). **The result shows that A_{up} can be modeled (i.e., the prediction accuracy of the model of A_{up} is 98%) when $i = n = 64$. However, if the inserted position is in the middle ($i = 32$) or at first stage ($i = 0$), then A_{up} can not be modeled (i.e., the prediction accuracy of the model of A_{up} is** 51%).

Since we cannot first build a model for A_{up} we must first try to build a model for A_{low}. This brings us to the second reason the (1,1)-iPUF is secure against the reliability based CMA-ES attack. Recall in Section 4.1 that Δ is needed to compute the fitness of each model $\mathbf{w}$. Δ is based on the input to A_{low}. However we do not know one of the input bits (the interposed bit) to A_{low} so we cannot compute Δ (see the calculation of Δ in Eq. (1) and (2)).

However, we should take a closer look at the way the computation of Δ is done to know how the interposed bit position affects the modeling attack on A_{low}. In Section 6.2, we have $\Delta = (1 - 2\mathbf{c}[j]) \times \Delta_{Flipping} + \Delta_{Non-Flipping}$ (see Eq.(13)) and thus, if j gets closer to 0, then Δ and $\Delta_{Non-Flipping}$ become similar. Strictly speaking, we cannot run CMA-ES to build a model for A_{low} when the interposed bit position i is **NOT** close to 0, for example the interposed bit position is in the range of $[n/2, n]$.

Conclusion: We cannot model A_{up} due to the unequal reliability information on the output and we cannot model A_{low} due to the unknown value of the interposed bit when the interposed bit position is properly chosen. Therefore, we claim the $(1, 1)$-iPUF is secure against the reliability based CMA-ES attack when the interposed bit position is properly chosen in the middle of the input to A_{low}.

Classical Machine Learning Attacks: The $(1, 1)$-iPUF is not secure against classical machine learning attacks due to its low model complexity. Instead of modeling the APUF components individually, any machine learning algorithm can be used to learn the model for A_{low} and A_{up} simultaneously. Experiments to support our claim are given in Section 8 (see Table 4). Note that, while the iPUF is vulnerable to classical derivative free machine learning, we will prove that derivative based classical machine attacks are **not** possible on an iPUF in the next section. As a result, we only need to consider derivative free classical machine learning attacks on an iPUF.

Attacks Using Linear Approximation (LA-CML and LA-RML): The security of the $(1, 1)$-iPUF against an ML attack that use the linear approximation (see Section 4.3) depends on the choice of the interposed bit position. In this attack, any CML or RML attack on an XOR PUF can be adapted to work on an (x, y)-iPUF. This adaptation is done by approximating the (x, y)-iPUF as a y-XOR APUF by fixing the interposed bit from the x-XOR APUF to be 0. In the case of the $(1, 1)$-iPUF this means that we ignore the component A_{up} and approximate $(1, 1)$-iPUF by APUF A_{low}.

Let us denote $\mathbf{c}_{low}$ as the input to A_{low} and $\mathbf{c}'_{low}$ to be the approximation of $\mathbf{c}_{low}$ where we fix the interposed bit r_{up} in $\mathbf{c}_{low}$ to be 0. We can write $A_{low}(\mathbf{c}'_{low})$ as the output of the linear approximation and $A_{low}(\mathbf{c}_{low})$ as the output of the $(1, 1)$-iPUF. We can now analyze how effective the linear approximation is. We measure the effectiveness of the approximation by computing the probability that the output of the linearized model

$A_{low}(\mathbf{c}'_{low})$ matches the output of $A_{low}(\mathbf{c}_{low})$:

$$p_{approx} = \Pr_{\mathbf{c}}(A_{low}(\mathbf{c}'_{low}) = A_{low}(\mathbf{c}_{low})) \tag{21}$$

If p_{approx} is high, then the $(1,1)$-iPUF **is accurately approximated** by APUF $A_{low}(\mathbf{c}'_{low})$ and can be modeled with an ML attack.

Let us assume the following conditions for the analysis of the attack and for the sake of explanation, we drop *low* from $\mathbf{c}'_{low}$ or $\mathbf{c}_{low}$. We model a $(x,1)$-iPUF with APUF components that are 100% reliable and the output r_{up} of the x-XOR APUF is uniform, i.e., $\Pr_{\mathbf{c}}(r_{up}=0) = \Pr_{\mathbf{c}}(r_{up}=1) = \frac{1}{2}$. Then,

$$\begin{aligned} p_{approx} &= \Pr_{\mathbf{c}}(A_{low}(\mathbf{c}') = A_{low}(\mathbf{c})) \\ &= \Pr_{\mathbf{c}}\left(A_{low}(\mathbf{c}') = A_{low}(\mathbf{c})|r_{up}=0\right)\Pr_{\mathbf{c}}(r_{up}=0) \\ &\quad + \Pr_{\mathbf{c}}(A_{low}(\mathbf{c}') = A_{low}(\mathbf{c})|r_{up}=1)\Pr_{\mathbf{c}}(r_{up}=1) \\ &= 1 \times 1/2 + \frac{n-i}{n} \times 1/2 = 1/2 + \frac{n-i}{2n}. \end{aligned} \tag{22}$$

Eq. (22) shows that p_{approx} decreases as i increases. **Note that our discussion holds for any $(x,1)$-iPUF and thus, it is applicable to the $(1,1)$-iPUF**. We model a 64-bit (1,1)-iPUF using the reliability based CMA-ES attack with the linear approximation (LA-RML). The number of challenge-reliability pairs is 30,000 and the noise rate is 20%. From the experiment, the prediction accuracy of LA-RML on a 64-bit (1,1)-iPUF when $i = 0, 32, 64$ is equal to 97%, 70% and 51%, respectively. Likewise, a similar result can be achieved by using CRP based CMA-ES (classical machine learning).

A prediction accuracy of 50% is the worst a machine learning attack can do on a PUF with uniform binary output. Therefore, it would seem that picking the interposed bit position to be as high as possible would result in the most secure (x,y)-iPUF design. However, below we will explain why choosing a high interposed position is not ideal.

Interposed Bit Position: In the $(1,1)$-iPUF the only design parameter we must choose is the interposed bit position. The higher the interposed bit position, the more influence A_{up} has on the $(1,1)$-iPUF. As a result the PUF model is more complex. It is more difficult to attack with CML attacks and the linear approximation is less accurate. However, a high bit position yields high noise on the output (less reliable). In addition, with a high interposed bit position and large enough number of input bits, the $(1,1)$-iPUF becomes equivalent to an XOR APUF in terms of susceptibility to RML attacks.

The conclusion from the analysis of the $(1,1)$-iPUF is that using the interposed bit position as the only security parameter is not enough to mitigate all the different ML attacks. We seek a trade-off between all the factors by choosing the interposed bit to be the middle bit position. Based on this choice we then analyze how to modify x and y to further secure our design in Section 6.5.

6.5 Security Analysis of the (x,y)-iPUF

The analysis in Section 6.4.2 shows that the $(1,1)$-iPUF with a middle interposed bit position can defeat the reliability based CMA-ES attack. However the $(1,1)$-iPUF is still vulnerable to CML attacks and both types of attacks that employ the linear approximation (LA-CML and LA-RML). In this section, we show how selecting x and y can mitigate the remaining ML attacks and we prove that the CRP based Logistic Regression attacks are not available and we show how the (x,y)-iPUF can mitigate the reliability based CMA-ES attack (RML), CML attacks (LR, DNN, CRP-based CMA-ES), LA-CML and LA-RML, and PAC learning attacks (Boolean function based attack, DFA attack and Perceptron attack).

Reliability based CMA-ES Attack: The security argument for the $(1,1)$-iPUF in Section 6.4.2 also applies to the (x,y)-iPUF. Using challenge-reliability pairs and CMA-ES, an adversary cannot build models for the x component APUFs in the x-XOR APUF. This is due to the unequal contribution of noise on the output between the x-XOR APUFs and the y-XOR APUFs (see Section 7.1). Also an adversary cannot build models for the y component APUFs in the y-XOR APUF as r_x (at the interposed bit position) is not known. Therefore, we consider the (x,y)-iPUF to be secure against the reliability based CMA-ES attack.

Classical Machine Learning Attacks: There is no known way the APUF components of the (x,y)-iPUF can be modeled individually. As a result, the only way to attack an (x,y)-iPUF is by modeling all the $(x+y)$ component APUFs simultaneously using classical machine learning, e.g., neural network or CRP based CMA-ES. Rührmair *et al.* [RSS+10] showed that the more APUFs that influence (contribute) to the output of an XOR APUF, the more difficult the PUF is to model using CML methods. In other words, increasing x in an x-XOR APUF can mitigate CML attacks. In Theorem 4, we prove that if the interpose position of r_x is i then the (x,y)-iPUF is equivalent to a $(y+p_r x)$-XOR APUF where:

$$p_r = \frac{1-(1-2p)^y}{2} \qquad \text{and} \qquad p = \frac{i}{n}.$$

If $i = n$ and y is odd, then $p_r = 1$ and (x,y)-iPUF is equivalent to $(x+y)$-XOR APUF in terms of security because of $(x+y)$ APUFs contributing to every challenge-response pair. If $i = 0$, then $p_r = 0$ and (x,y)-iPUF is equivalent to a y-XOR PUF. In terms of difficulty of modeling with classical machine learning methods (i.e. CRP based CMA-ES), the (x,y)-iPUF with parameter i is approximately equivalent to a $(y+p_r x)$-XOR PUF. We experimentally verify this claim in Section 8.2 (see Figs. 2 and 5). As a result, we can use the same strategy employed in the XOR APUF design [RSS+10] and increase x and y in an (x,y)-iPUF to mitigate classical machine learning attacks. It is also worth noting this analysis to determine the number of APUFs that contribute in an iPUF can be used to derive further iPUF properties such as uniformity, uniqueness and reliability.

LA-RML. Reliability based CMA-ES applied to an (x,y)-iPUF after linearly approximating it as a y-XOR APUF learns one single component APUF of the y-XOR APUF. In the linear approximation we substitute r_x from the upper x-XOR APUF by 0 and feed 0 into the lower y-XOR APUF. If we denote the single component APUF which we try to learn using CMA-ES by A_{low}, then, for the challenge interposed with r_x, the output of A_{low} is the output of an $(x,1)$-iPUF. I.e., we attempt to learn a model for A_{low} from a linear approximation of an $(x,1)$-iPUF. From Eq.(22) with interpose bit position in the middle we infer that the model at best predicts A_{low} with $p_{approx} = 75\%$ accuracy. Reliability based CMA-ES learns models for each of the component APUFs of the lower y-XOR APUF, each with at most 75% accuracy. This shows that the maximum accuracy of the learned approximated y-XOR APUF is at most

$$\begin{aligned} p_{learn}^{XOR} &= \sum_{j=0,j \text{ is even}}^{y} \binom{y}{j} (1-p_{approx})^j p_{approx}^{y-j} \\ &= \frac{1+(2p_{approx}-1)^y}{2} = \frac{1}{2} + \frac{1}{2^{y+1}}. \end{aligned}$$

This shows that p_{learn}^{XOR} is close to $1/2$ for y large enough and this implies that the learned model for the linearly approximated y-XOR APUF does not contain predictive value[5].

[5] We can compare the model's output with the iPUF's output and attempt to find out whether this

If $y \geq 3$, then $p_{learn}^{XOR} \leq 56.25\%$. We launched the LA-RML attack on a 64-bit (3,3)-iPUF. In this experiment the noise rate for each APUF was 20%. The prediction accuracy of the final model was around 50% (confirming the theoretical upper bound) when 200,000 challenge-reliability pairs were used in the training phase (see Table 4).

LA-CML. In case of linearly approximating the (x, y)-iPUF as a y-XOR APUF and applying classical ML, we know that learning a model for even a noise-free 64 bit y-XOR APUF is currently not practical for $y \geq 10$ if LR is used and not practical for even smaller y if CRP based CMA-ES is used. However, even if the iPUF itself is 100% reliable, a *reliable* CRP of the iPUF may be a *noisy* CRP of the linearly approximated y-XOR APUF. By combining Eq.(15) with the derivation leading to Eq.(22), the accuracy p_{approx}^{XOR} of the linearly approximated y-XOR APUF itself is given by

$$p_{approx}^{XOR} = 1 \times 1/2 + p' \times 1/2,$$

where

$$p' = \sum_{j=0, j \text{ is even}}^{y} \binom{y}{j} \left(\frac{i}{n}\right)^j \left(\frac{n-i}{n}\right)^{y-j} = \frac{1 + (1 - \frac{2i}{n})^y}{2}.$$

After substituting p' we obtain

$$p_{approx}^{XOR} = \frac{3}{4} + \frac{1}{4}(1 - \frac{2i}{n})^y.$$

If the interposed bit is at the middle position (i.e. $i = \frac{n}{2}$), then p_{approx}^{XOR} is 75%. This implies that the noise rate of the linearly approximated y-XOR APUF given by the CRPs from the iPUF is 25%. For this reason CML should be even more difficult: We performed the LA-CML attack using CRP based CMA-ES on a **reliable** 64-bit (3,3)-iPUF. The prediction accuracy of the model is around 50% when 200,000 CRPs are used in the training phase (see Table 4).

Logistic Regression Attack on the iPUF: In the analysis of the security of the iPUF against classical ML attacks, we **do not specifically** consider the type of machine learning attack, i.e., deep neural network, Logistic Regression, or CRP based CMA-ES. We prove that the Logistic Regression attacks can at best learn a linear approximated model of the iPUF and therefore reduce to LA-RML or LA-CML.

Basically, CRP based Logistic Regression works more efficiently than CRP based CMA-ES when the searching space is large. For example, in [TB15], using Logistic Regression we can successfully model 4-XOR APUF with 15,000 CRPs only while using CMA-ES we cannot have a good model for 4-XOR APUF with 200,000 CRPs (see Figure 2). Hence, it is important to know if there exists any possible Logistic Regression based attacks on the iPUF or not. We show that the answer to this question is **NO** by analyzing Logistic Regression (LR).

In the upper x-XOR APUF of the iPUF, since there are x n-bit APUF instances, we denote $\mathbf{w}^x = (\mathbf{w}_1^x, \ldots, \mathbf{w}_x^x)$ as the model of the x-XOR APUF. Further we denote $\mathbf{w}_i^x = (\mathbf{w}_i^x[0], \ldots, \mathbf{w}_i^x[n])$ as the $(n+1)$ dimensional vectors and the models of the APUFs in the x-XOR APUF, $i = 1, \ldots, x$. Similarly, $\mathbf{w}^y = (\mathbf{w}_1^y, \ldots, \mathbf{w}_y^y)$ is the model of the y-XOR APUF of the iPUF and $\mathbf{w}_i^y$ are $(n+2)$ dimensional vectors and the models of APUFs in the y-XOR APUF. In order to enable derivative based ML attacks, we follow the approach proposed in [RSS+10, Söl09] (this is the Logistic Regression ML attack on an x-XOR APUF), i.e. we approximate the discrete output r_x and r_y by a continuous

teaches something about r_x. Due to the XOR in the lower y-XOR APUF, the outputs of individual component APUFs are masked so that little can be learned about r_x. For small $y = 1$, we are able to learn some information about r_x from comparing the model's output with the iPUF's actual output; this can then be used to learn about the upper x-XOR APUF.

function sigmoid $\sigma(\cdot)$ where $\sigma(x) = \frac{1}{1+\exp(-x)}$. More precisely, we define the following functions (here $\mathbf{\Psi}(\mathbf{c}, \hat{r}_x)$ denotes $\mathbf{\Psi}(.)$ applied to challenge $\mathbf{c}$ interposed with $\hat{r}_x$):

$$\begin{aligned}
\Delta_x &= g_x(\mathbf{w}^x, \mathbf{c}) = \prod_{i=1}^{x} \langle \mathbf{w}_i^x, \mathbf{\Psi}(\mathbf{c}) \rangle, \\
r_x &= \delta(\Delta_x) = \delta(g_x(\mathbf{w}^x, \mathbf{c})), \\
\hat{r}_x &= \delta(\Delta_x) + e(\Delta_x) = \delta(g_x(\mathbf{w}^x, \mathbf{c})) + e(g_x(\mathbf{w}^x, \mathbf{c})), \\
\Delta_y &= g_y(\mathbf{w}^y, \mathbf{c}, \hat{r}_x) = \prod_{i=1}^{y} \langle \mathbf{w}_i^y, \mathbf{\Psi}(\mathbf{c}, \hat{r}_x) \rangle, \\
\hat{r}_y &= \sigma(\Delta_y) = \sigma(g_y(\mathbf{w}^x, \mathbf{c}, \hat{r}_y)),
\end{aligned}$$

where $\delta(x)$ is the step function, i.e., $\delta(x) = 0$ if $x > 0$, and $\delta(x) = 0$ otherwise, and e is a certain error function chosen by the adversary. The function δ has derivative of 0 everywhere except $x = 0$ (where the derivative is ∞) and $e(x)$ has derivative everywhere. Since in practice Δ_x is never exactly equal to 0, we may assume that the derivative of δ in Δ_x is always equal to 0.

In order to find the optimal solution of $\mathbf{w} = (\mathbf{w}^x, \mathbf{w}^y)$ (i.e. the model for the (x, y)-iPUF) from a randomly generated model $\mathbf{w}$, we define the following function as described in [RSS+10, Söl09]:

$$l = -\frac{1}{N} \sum_{(\mathbf{c}_i, r_i), i=1,\ldots,N} \ln(\sigma(\Delta_y)^{r_i}(1 - \sigma(\Delta_y))^{1-r_i})$$

where $\{(\mathbf{c}_1, r_1), \ldots, (\mathbf{c}_N, r_N)\}$ are the challenge-response pairs of the iPUF in the training set. After that, we need to compute the gradient of l in order to find the optimal solution, i.e., we need to compute

$$\begin{aligned}
\nabla l = (&\frac{\partial l}{\partial \mathbf{w}_1^x[0]}, \ldots, \frac{\partial l}{\partial \mathbf{w}_1^x[n]}, \ldots, \frac{\partial l}{\partial \mathbf{w}_x^x[0]}, \ldots, \frac{\partial l}{\partial \mathbf{w}_x^x[n]}, \\
&\frac{\partial l}{\partial \mathbf{w}_1^y[0]}, \ldots, \frac{\partial l}{\partial \mathbf{w}_1^y[n]}, \ldots, \frac{\partial l}{\partial \mathbf{w}_y^y[0]}, \ldots, \frac{\partial l}{\partial \mathbf{w}_y^y[n]})
\end{aligned}$$

After that we will update $\mathbf{w} = \mathbf{w} - \eta \nabla l$ where η is the learning stepsize. By updating like this many times, we hope that the algorithm will converge to an optimal solution $\mathbf{w}^*$. Now, we focus on the calculation $\frac{\partial l}{\partial \mathbf{w}_i^x[j]}$ which is equal to:

$$\begin{aligned}
&= \partial(-\frac{1}{N} \sum_{(\mathbf{c}_i, r_i), i=1,\ldots,N} \ln(\sigma(\Delta_y)^{r_i}(1 - \sigma(\Delta_y))^{1-r_i}))/\partial \mathbf{w}_i^x[j] \\
&= -\frac{1}{N} \sum_{(\mathbf{c}_i, r_i), i=1,\ldots,N} [r_i(1 - \sigma(\Delta_y)) - (1 - r_i)\sigma(\Delta_y)] \frac{\partial \Delta_y}{\partial \mathbf{w}_i^x[j]}.
\end{aligned}$$

We have

$$\begin{aligned}
\frac{\partial \Delta_y}{\partial \mathbf{w}_i^x[j]} &= \frac{\partial g_y}{\partial \mathbf{w}_i^x[j]} = \frac{\partial g_y}{\partial [\delta(\Delta_x) + e(\Delta_x)]} \frac{\partial [\delta(\Delta_x) + e(\Delta_x)]}{\partial \mathbf{w}_i^x[j]} \\
&= \frac{\partial g_y}{\partial [\delta(\Delta_x) + e(\Delta_x)]} \frac{\partial e(\Delta_x)}{\partial \Delta_x} \frac{\partial \Delta_x}{\partial \mathbf{w}_i^x[j]}
\end{aligned}$$

(since $\delta'(\Delta_x) = 0$ as explained above).

But if we consider using the linear approximation with this attack where we fix $\hat{r}_x = 0 + e(\Delta_x)$, then we also have the same result, i.e.,

$$\frac{\partial \Delta_y}{\partial \mathbf{w}_i^x[j]} = \frac{\partial g_y}{\partial [\delta(\Delta_x) + e(\Delta_x)]} \frac{\partial e(\Delta_x)}{\partial \Delta_x} \frac{\partial \Delta_x}{\partial \mathbf{w}_i^x[j]},$$

since Δ_x is exactly equal to 0 with probability 0 and outside $\Delta_x = 0$, $\delta(\Delta_x)$ has derivative 0.

This fact implies that we cannot distinguish the partial derivative of the iPUF from that of the linear approximated model of the iPUF. From this analysis, we conclude that Logistic Regression attacks are equivalent to Logistic Regression attacks that use the linear approximation. Due to this fact, the iPUF can mitigate Logistic Regression attacks just like it can mitigate attacks that use the linear approximation by choosing $y \geq 3$.

We already proved that the (x, y)-iPUF with the interposed bit position in the middle is equivalent to a $(y + \frac{x}{2})$-XOR APUF in terms of general security. From Inequality (6), we can now argue that $y + \frac{x}{2} < 10$ in order to defeat derivative free modeling attacks on a $(y + \frac{x}{2})$-XOR APUF: The number 10 is reported in [TB15], where it is shown that if $y + \frac{x}{2} = 10$, then the state-of-the art derivative based machine learning attacks are infeasible on a $(y + \frac{x}{2})$-XOR APUF. This means that iPUFs require less APUF components than an XOR APUF to be secure. As a result, iPUFs are better in terms of security, reliability and hardware overhead compared to XOR APUFs.

Deep Neural Network Attacks on the iPUF: To the best of our knowledge, there is no theory which can tell us the lower bound of the number of required CRPs for using deep neural networks to attack iPUFs. Instead, we leverage the relationship between XOR APUFs and iPUFs we previously developed to map this problem to attacking XOR APUFs using deep neural networks. In this attack, the deep neural networks need to learn the structure of the XOR APUFs (no specific mathematical model is assumed). Essentially, this means deep neural networks use a **black box learning** approach. However, Logistic Regression use a precise mathematical model of the XOR APUF. This means LR is more powerful for attacking XOR APUFs as compared to deep neural networks. Hence, if we use choose x, y such that $(y + \frac{x}{2})$-XOR APUF is secure against Logistic Regression attacks, then we know that the (x, y)-iPUF with the interpose bit in the middle is also secure against deep neural network attacks. For example, we can choose $x = 2$, $y = 9$ (such that $y + \frac{x}{2} = 10$), interpose bit $i = 32$, and challenge length $n = 64$, knowing that 64-bit 10-XOR APUF is secure against Logistic Regression attack in practice [TB15]. The detailed experimental results for deep neural network attacks on XOR APUFs can be found in Section 8.2.2.

(CRP-based) CMA-ES Attacks on the iPUF: CMA-ES as a derivative-free optimization method, is always less efficient than a derivative-based optimization method, like Logistic Regression. This is because the derivative tells the best direction towards the optimal solution. So far there is no theory for the lower bound on the number of required CRPs for using CMA-ES to model iPUFs or XOR APUFs. Hence, we use the same methodology as we discuss for deep neural networks above. Furthermore, comparing CRP-based CMA-ES to Logistic Regression, experimentally (see Section 8) we can see that in general Logistic Regression can model an XOR APUF with higher accuracy using less CRPs than CRP-based CMA-ES. Therefore, the (x, y)-iPUF can be secured against the CRP-based CMA-ES attacks by setting x, y such that $(y + \frac{x}{2})$-XOR APUFs are resilient against Logistic Regression.

PAC-Learning Attacks on the iPUF: In [GTS15, GTS16, GTFS16], the authors proposed two novel modeling attacks for the PAC learning problem. PAC learning attacks can be divided into three categories: Boolean functions based attack [GTFS16], Perceptron attack [GTS15] and Deterministic Finite Automata (DFA) attack [GTS16].

The attack described in [GTFS16] does not work for the iPUF because of the following reason. As described in [GTFS16], assume that if the PUF has k number of *influential bits* (or average sensitivity) (see Section 2.2 and Theorem 3 in [GTFS16] for the detailed definition), then this PUF can be ϵ-approximated by another Boolean function h depending

on only a constant number of Boolean variables K, where

$$K = e^{\frac{k}{\epsilon}\times(2+\sqrt{\frac{2\epsilon\log_2(4k/\epsilon)}{k}})} > e^{\frac{2k}{\epsilon}}.$$

We notice the definition of *influence of variable i* in Section 2.2 in [GTFS16] is equivalent to the definition of SAC in Section 7. As shown in [GTFS16], for a n-bit (x, y)-iPUF with interposed position j, k can be computed as follows:

$$k = \sum_{i=0,i\neq j}^{n} p_i \overset{Eq.(28)}{=} \sum_{i=0,i\neq j}^{n} \{\frac{1+(1-2\frac{i}{n})^x}{2}\frac{1-(1-2\frac{i}{n+1})^y}{2} + \frac{1-(1-2\frac{i}{n})^x}{2}\frac{1-(1-2\frac{|i-j|}{n})^y}{2}\}.$$

For 64-bit $(3, 3)$-iPUF with interpose position $j = 31$, $k = 25.2$. Hence, even if we consider a very weak approximated function h with $\epsilon = 0.5$, then $K = e^{25.2\times2/0.5} > e^{100} > 2^{100}$. However, Theorem 3 in [GTFS16] tells us that if K is small then we can model the PUF and it does not confirm that we cannot approximate the iPUF by a Boolean function. To show the resistance against the attack described in Theorem 4 in [GTFS16], we need to do one more step, i.e., we apply $\tilde{k}$-junta testing [Bea] for our iPUF. The $\tilde{k}$-junta testing we implemented follows [Bea] and it is fully explained in Appendix A.4. We applied this test[6] with $\tilde{k} = 63$ and $\epsilon = 0.2$ to the most simple version of an iPUF, i.e., 64-bit (1,1)-iPUF, in simulation. Since $\tilde{k}$-junta testing is a randomized test with a certain correct probability, we ran the test for 100 times. It consistently outputs that it is not a 63-junta after 100 runs. Let us analyze our confidence in this result below.

Let $1 - p$ denote the probability such that the outcome O of a $\tilde{k}$-junta test correctly matches the ground truth T. Suppose we perform m times a k-junta test leading to an output pattern with $m - y$ "No" and y "Yes" outcomes. Since ground truth is either all outcomes are "No" or all outcomes are "Yes", we know that, conditioned on the output pattern, the $\tilde{k}$-junta tests either correctly matches ground truth T a total of $m - y$ times (where $O = T$ equals "No") and incorrectly matches T a total of y times (where O equals "Yes") or correctly matches ground truth T a total of y times (where $O = T$ equals "Yes") and incorrectly matches T a total of $m - y$ times (where O equals "No"). This gives

$$Pr(T = \text{"No"}|\text{output pattern}) = \frac{p^y(1-p)^{m-y}}{p^y(1-p)^{m-y} + p^{m-y}(1-p)^y} = \frac{1}{1+\left(\frac{p}{1-p}\right)^{m-2y}}.$$

Since $\tilde{k}$-junta testing has the property $p \leq 1/3$, we have $p/(1-p) \leq 1/2$, hence,

$$Pr(T = \text{"No"}|\text{output pattern}) \geq \frac{1}{1+2^{-(m-2y)}} \geq 1 - 2^{-(m-2y)}.$$

In our execution of $\tilde{k}$-junta testing we used $m = 100$ and observed $y = 0$. Conditioned on this observation we conclude that T equals "No" with probability $\geq 1 - 2^{-100}$ based on the derived formula.

This result can be explained intuitively because the influence of any challenge bit i (or SAC property of i) is larger than 0 in an iPUF (see Section 7.2 for theoretical analysis of SAC of iPUF, and see Figure 6 for its experimental validation). In other words, it means that every challenge bit has a contribution to the final response of an iPUF. Since the computed $\tilde{k} = 64$, the time complexity for the attack used in [GTFS16] on

[6] ϵ measures the modeling inaccuracy of the given function by any $\tilde{k}$-junta, and it is taken from the range between 0 and 1. By setting $\epsilon = 0.2$, in the $\tilde{k}$-junta test, we are testing whether there exists a $\tilde{k}$-junta that can model the given function with at least 80% accuracy.

64-bit (1,1)-iPUF will be $\mathcal{O}(64^{64})$ which makes the attack infeasible (see the discussion in Theorem 4 in [GTFS16]). Moreover, we performed $\tilde{k}$-junta testing with the same setup for (1,1)-iPUF on 64-bit 2-XOR APUF and APUF, and the results turn out to be 64, which is consistent with the SAC property analysis of XOR APUFs and APUFs in [MKP09, DGSV14, NSCM16].

If we compare the LR attack on XOR APUF in [RSS+10, TB15] with the Perceptron attack on XOR APUF in [GTS15] or with the DFA attack on XOR APUF in [GTS16] (DFA attack is demonstrated on APUFs, but [GTS16] also claims that it is possible to attack XOR APUFs with some modifications), we can see that the LR algorithm is much more powerful. The reasons are LR algorithm has a much smaller searching space, uses the XOR APUF mathematical model, and uses information of the gradient in an optimization process. As shown in Section 6.3, the (x, y)-iPUF is equivalent to the $(y + x/2)$-XOR APUF under classical ML. Therefore, the attacks in [GTS15, GTS16] do not work for the iPUF when x and y are properly chosen.

7 Reliability Analysis and Strict Avalanche Criterion of (x, y)-iPUF

7.1 Reliability of the (x, y)-iPUF

We develop a formula for computing the noise of the x-XOR APUF where the noise rate of all APUFs are the same, i.e., all APUFs have a noise rate of $\beta, 0 \leq \beta \leq 1$. Let β_x be the noise rate of the x-XOR APUF. For a given challenge $\mathbf{c}$, if there is an odd number of noisy APUF responses, then XOR APUF's output is noisy. Hence,

$$\begin{aligned}\beta_x &= \sum_{j=0, j \text{ is odd}}^{x} \binom{x}{j} \beta^j (1-\beta)^{x-j} \\ &= \frac{1-(1-2\beta)^x}{2}. \end{aligned} \tag{23}$$

Now, we compute the unreliability (i.e. the noise) of the (x, y)-iPUF. We consider the following two cases.

Case I: The output of the x-XOR APUF is reliable. This event occurs with probability $1 - \beta_x$, in which case the noise rate of the output of the iPUF denoted as β_I is equal to the noise rate of the y-XOR APUF denoted as β_y, i.e.,

$$\beta_I = \beta_y = \frac{1-(1-2\beta)^y}{2}. \tag{24}$$

Case II: The output of the x-XOR APUF is unreliable. This event occurs with probability β_x. In this case (see Section 6.4.1), each APUF in the y-XOR APUF will have noise $\beta' = \beta(1 - \frac{i}{n}) + (1 - \beta)\frac{i}{n} = \beta + (1 - 2\beta)\frac{i}{n}$. Hence in this case, the noise rate of the iPUF, denoted as β_{II}, is equal to the noise rate of the y-XOR APUF where all APUFs have the noise rate β', i.e.,

$$\begin{aligned}\beta_{II} &= \frac{1-(1-2\beta')^y}{2} = \frac{1-(1-2(\beta+(1-2\beta)\frac{i}{n}))^y}{2} \\ &= \frac{1-(1-2\beta)^y(1-2\frac{i}{n})^y}{2} \overset{Eq.(24)}{=} \frac{1-(1-2\beta_y)(1-2\frac{i}{n})^y}{2}. \end{aligned} \tag{25}$$

Therefore, the noise rate of the (x, y)-iPUF, denoted as β_{iPUF}, is equal to

$$\begin{aligned}\beta_{iPUF} &= (1-\beta_x)\beta_I + \beta_x\beta_{II} = \beta_y + \beta_x(\beta_{II} - \beta_y) \\ &= \beta_y + \beta_x(\frac{1-(1-2\beta_y)(1-2\frac{i}{n})^y}{2} - \beta_y) \\ &= \beta_y + \frac{\beta_x}{2}(1-2\beta_y)(1-(1-\frac{2i}{n})^y). \end{aligned} \tag{26}$$

The following examples confirm the correctness of Eq.(26). If $i = 0$, then we have $\beta_{iPUF} = \beta_y$. When $i = 0$, the (x, y)-iPUF is basically equivalent to the y-XOR APUF. Therefore, the calculated result $\beta_{iPUF} = \beta_y$ is confirmed.

If $i = n$ and y is odd, then from Eq.(26), $\beta_{iPUF} = \beta_x + \beta_y - 2\beta_x\beta_y = \beta_x(1-\beta_y) + \beta_y(1-\beta_x) = \beta_{(x+y)-\text{XOR APUF}}$. We know that when $i = n$ and y is odd, the (x, y)-iPUF is exactly the $(x+y)$-XOR APUF. Hence, the calculated result $\beta_{iPUF} = \beta_{(x+y)-\text{XOR APUF}}$ is confirmed.

7.2 Strict Avalanche Criterion of (x, y)-iPUF

The SAC property is an important security feature as discussed in [MKP09, DGSV14, NSCM16]. Here we analyze the SAC property of the (x, y)-iPUF. Assume that the output r_x of the x-XOR APUF is interposed at position j in the challenge to the y-XOR APUF, $j = 0, 1, \ldots, n$. We would like to compute the probability that flipping a bit in the input results in the flipping of the output bit of the (x, y)-iPUF. This analysis for the (x, y)-iPUF is similar to the analysis we did for the APUF in Section 6.2:

$$p_i = \Pr_{\mathbf{c}}(r_{c[i]=0} \neq r_{c[i]=1}), i = 0, 1, \ldots, n-1 \tag{27}$$

where $r_{c[i]=0}$ and $r_{c[i]=1}$ are the output of the (x, y)-iPUF when bit $\mathbf{c}[i] = 0$ and $\mathbf{c}[i] = 1$, respectively, while all the other challenge bits are same. To compute p_i, we consider the following two cases:

Case I. $\mathbf{c}[i]$ flips, r_x does not flip, r flips. In this case, we flip challenge bit $\mathbf{c}[i]$ but the output r_x of the x-XOR APUF does not flip and the (x, y)-iPUF's output r flips (i.e., $r_{c[i]=0} \neq r_{c[i]=1}$). From the analysis in Section 6.2, we know that if the challenge bit $\mathbf{c}[i]$ flips, then the output of an APUF in an x-XOR APUF will flip with expected probability $p = \frac{i}{n}$ and thus, the output of x-XOR APUF will flip with an expected probability $p_x = \frac{1-(1-2p)^x}{2}$. In other words, r_x will not flip with a probability $\bar{p}_x = 1 - p_x = \frac{1+(1-2p)^x}{2}$. When r_x is not flipped due to $\mathbf{c}[i]$ flipping, the output of an APUF in the y-XOR APUF will be flipped by flipping $\mathbf{c}[i]$ with a probability $p' = \frac{i}{n+1}$. Thus the probability that r_x does not flip (and r flips) when flipping $\mathbf{c}[i]$ is equal to:

$$p_I = \frac{1+(1-2p)^x}{2} \cdot \frac{1-(1-2p')^y}{2}.$$

Case II. $\mathbf{c}[i]$ flips, r_x flips and r flips. If r_x flips because $\mathbf{c}[i]$ flips, then there are two flipping challenge bits at position i and j in the input challenge to the y-XOR APUF. In this case, the output of an APUF in the y-XOR APUF will be flipped with a probability $p'' = \frac{|i-j|}{n+1}$. The computation for p'' is beyond the scope of this paper but a detailed derivation of it can be found in [NSCM16]. Thus the output r will flip with a probability

$$p_{II} = \frac{1-(1-2p)^x}{2} \cdot \frac{1-(1-2p'')^y}{2}$$

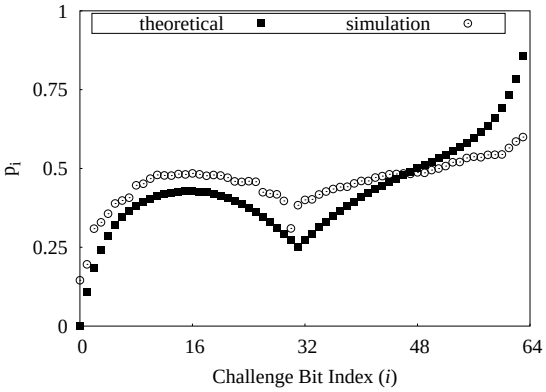


Figure 3: SAC property of 64-bit (3,3)-iPUF with theoretical and simulation data.

Therefore, we have

$$\begin{aligned} p_i =& p_I + p_{II} \\ =& \frac{1+(1-2p)^x}{2} \cdot \frac{1-(1-2p')^y}{2} \\ &+ \frac{1-(1-2p)^x}{2} \cdot \frac{1-(1-2p'')^y}{2} \\ =& \frac{1+(1-2\frac{i}{n})^x}{2} \cdot \frac{1-(1-2\frac{i}{n+1})^y}{2} \\ &+ \frac{1-(1-2\frac{i}{n})^x}{2} \cdot \frac{1-(1-2\frac{|i-j|}{n})^y}{2} \end{aligned} \tag{28}$$

We experimentally verify our calculations for the SAC property of the $(3,3)$-iPUF and the result is described in Fig 3. The simulation of the SAC property is computed by using 20,000 pairs of CRP for computing each p_i.

8 Simulation Results

We provide simulation results in this section to support the major security and reliability claims made regarding the iPUF.

8.1 Experimental Verification of the Classical Machine Learning Hierarchy

In this section we experimentally verify Theorem 1 regarding the efficiency of different classical machine learning attacks. Recall that we stated LR is more efficient in attacking an XOR APUF than CMA-ES and Deep Neural Networks. To provide empirical evidences of this we ran the following experiment: We ran these three attacks on a 4-XOR APUF with various numbers of CRPs as shown in Table 3.

We ran CRP-based CMA-ES with 200,000 CRPs (Fig. 2) and 500,000,000 CRPs (Section 8.2.2). Regardless of the two possible CRPs numbers used, CRP based CMA-ES had a prediction accuracy of around 50%, clearly showing that the attack fails. When we use a deep neural network with 30,000 CRPs the highest accuracy we can achieve after 10 runs is 94% which is significantly better than CRP based CMA-ES with a smaller number of CRPs. Lastly in Table 3 we run the LR attack which is the most efficient of all the classical attacks tested here. LR attacks requires only 12,000 CRPs to achieve a

Table 3: Prediction accuracy of different algorithms using the same amount of CRPs for training.

#CRPs	LR	DNN	CMA-ES
12,000	98%	55%	50%
30,000	99%	94%	50%

Table 4: Vulnerability of different PUF designs to machine learning attacks. CML=Classical Machine Learning attack, RML=Reliability based Machine Learning attack, LA-RML=Linear Approximation Reliability based Machine Learning attack, LA-CML=Linear Approximation Classical Machine Learning attack.

		PUF Design		
		6-XOR APUF	(1,1)-iPUF	(3,3)-iPUF
Attack	CML	50.2% (✗) †	82.4% (✓)‡	52.9% (✗)
	RML	84.0% (✓)	N/A(✗)	N/A (✗)
	LA-CML	N/A(✗)	74.0% (✓)	49.0% (✗)
	LA-RML	N/A(✗)	73.0% (✓)	48.0% (✗)

† Attack fails
‡ Attack works

prediction accuracy of around 99% [RSS+10]. Thus, from these experimental results we can empirically show that LR is the most efficient attack on XOR APUFs, followed by deep neural networks. The least efficient classical machine learning attack is CRP based CMA-ES.

8.2 iPUF Security

8.2.1 Simulated Machine Learning Attacks On iPUF and XOR APUF

To compare the vulnerabilities of the XOR APUF and various iPUF configurations to machine learning attacks we used the following setup: We simulated a 64-bit 6-XOR APUF, a 64-bit $(1,1)$-iPUF and a 64-bit $(3,3)$-iPUF using Matlab. Note that, all the iPUFs have the interposed bit in the middle position.

The APUF components that make up each PUF design use weights that follow a normal distribution with $\mu = 0$ and $\sigma = 0.05$. The noise for each weight follows a normal distribution $\mathcal{N}(0, \sigma^2_{\text{noise}})$. The distribution of each weight is therefore $\mathcal{N}(0, \sigma^2 + \sigma^2_{\text{noise}})$. In our simulations, we used the following relation between σ and σ_{noise} to control the reliability levels: $\sigma_{\text{noise}} = \gamma\sigma$, where $0 \leq \gamma \leq 1$. For $\gamma = 0$, a PUF instance is 100% reliable. Note that this model does not include the noise introduced by the arbiter circuit itself, but it still properly serves our analysis purpose. The reason is that in practice, the hardware footprint of a switch and an arbiter are roughly same. Hence, the noise introduced by an arbiter circuit is very small compared with all the noises introduced by all switches in an APUF because there are multiple switches but only one arbiter in an APUF. Also, this simulation method is widely used in literature and it has been have observed that this method matches what occurs in practice [Bec15, DV13].

On each respective PUF design we run three different machine learning attacks (two in the case of the XOR APUF). We perform a classical machine learning attack (denoted as CML in Table 4), the reliability based CMA-ES attack (denoted as RML in Table 4) and the linear approximation to the classical and reliability based machine learning attack (denoted as LA-CML and LA-RML in Table 4).

Each attack in Table 4 is performed using CMA-ES to optimize the mathematical model for 1000 iterations. Each attack uses 200,000 CRPs for training (or 200,000 challenge reliability pairs in the case of the reliability attacks). The accuracy of the attack is computed based on a testing dataset of 2000 CRPs. We compute the accuracies reported in Table 4 by running each attack 10 times and taking the average. Note the accuracy of

best-working attacks is not much different from the average one (1% or 2% difference).

The security of the 6-XOR APUF is shown in Table 4 in the first column. It is clear that an XOR APUF using a large number of APUF components (in this case 6) can mitigate classical machine learning attacks, given the amount of training data provided in this setup. However, the 6-XOR APUF is still highly vulnerable to reliability-based ML attacks. This result is demonstrated in Table 4 as the reliability based CMA-ES attack achieves an average accuracy of 84% on this PUF design. Note that we do not run any attacks on the 6-XOR APUF that use the linear approximation because these types of attacks are specific to the iPUF.

The resilience of the $(1,1)$-iPUF to classical machine learning, LA-CML and LA-RML is shown in column two of Table 4. We note the following: As hypothesized, the $(1,1)$-iPUF is vulnerable to classical machine learning due to its low model complexity. However, the linear approximation reliability based CMA-ES attack (LA-RML) works on the $(1,1)$-iPUF which may seem unexpected given our previous claims. Recall that in general the reliability based CMA-ES attack cannot be performed on any iPUF design (hence the X in entry for the reliability-based ML attack for the $(1,1)$-iPUF). This is because the input to the y-XOR APUF is unknown and therefore Δ cannot be computed. However, in our LA-RML attack on the $(1,1)$-iPUF and $(3,3)$-iPUF we do not use the original formulation of the reliability based CMA-ES attack. To make the attack work on iPUF configurations we assume the interposed bit to be 0. By doing this we can now compute Δ and treat the $(1,1)$-iPUF as a single APUF. In section 6.5 we showed that for the $(x,1)$-iPUF using any ML technique with the linear approximation would have an upper bounded model accuracy of 75%. The model accuracy produced by the LA-RML and LA-CML on the $(1,1)$-iPUF in Table 4 closely matches our theoretical upper bound.

The security of the $(3,3)$-iPUF against classical machine learning, LA-CML and LA-RML is shown in the third column of Table 4. It can clearly be seen that all attacks fail to produce an accurate model of the $(3,3)$-iPUF. Due to the high model complexity, the $(3,3)$-iPUF is not vulnerable to the classical machine learning attack, just like the 6-XOR APUF. Just like for the $(1,1)$-iPUF, the reliability based CMA-ES attack cannot be performed on the $(3,3)$-iPUF. When we run the linear approximation reliability based CMA-ES attack, due to the higher y, the accuracy of the model generated by any method that uses the linear approximation is theoretically upper bounded at 75% (See Eq. (22)). This coincides with the low accuracy of the model generated by both the linear approximation reliability based CMA-ES attack and the linear approximation CRP based CMA-ES. Overall our experimental results for this section support our claim that the (x,y)-iPUF with proper parameter choices is secure against all current state-of-the-art of machine learning attacks.

8.2.2 Some Notes on x-XOR APUF

Under state-of-the-art classical ML, the x-XOR APUF must resist Logistic Regression attacks, deep neural network attacks and CRP-based CMA-ES attacks. However, the iPUF only needs to mitigate deep neural network attacks and CRP-based CMA-ES attacks (since Logistic Regression attack is not possible). In this section we present experiments to determine $y + x/2$ in an $(y + x/2)$-XOR that resists both aforementioned attacks. Based on these results we can use the equivalence proof developed earlier to determine the corresponding (x,y)-iPUF parameters.

We ran CRP-based CMA-ES attacks and deep neural network attack experiments to determine the minimum value of $y + x/2$ to create a secure PUF design. We ran CRP based CMA-ES for 2,000 iterations using 500,000,000 CRPs for training and 2,000 CRPs for testing. In this experiment we run a parallelized version of CMA-ES in C# using an Intel Xeon E5-2680 14 core machine with 128 GB of RAM. In this setup the maximum number of CRPs we can use is limited by the RAM (storing the CRPs in a byte data array in C#). On a 128 bit 4-XOR APUF the average accuracy for 10 attack runs on the testing

Table 5: Modeling accuracy using a Deep Neural Network

XOR number	Bit Number		
	64	128	256
5	0.987	0.986	0.975
6	0.976	0.977	0.533
7	0.976	0.491	-
8	0.513	-	-
9	0.502	-	-

– The result is not available.

set was 51.77%.

In deep neural network attacks, the adversary uses the challenge transformation mapping (see Eq. (1)) and a deep neural network for the attack (contrary to the pure black box attack which only employs the deep neural network without the challenge transformation mapping). The first study on how to implement deep neural network attacks on APUF based designs including iPUF (for small x and y) was done by Pranesh et.al [SBC19]. For our deep neural network experiments we created a feed forward neural network. Our neural network structure is as follows: 3 hidden layers with 100, 60 and 20 neurons respectively, and a Softmax layer on the output. For the neurons in the hidden layers, we used a sigmoid activation function. Our network is trained by minimizing the cross-entropy loss function using the optimization technique ADAM [KB14]. We trained our network for 20 epochs using 2,000,000 CRPs (with a batch size of 128) and use 2,000 CRPs for testing.

In our computational setup we use an Intel Xeon E5-2680 14 core machine to generate the CRPs and corresponding responses in parallel in C#. We then load this data into TensorFlow in Python and train the neural network using an Nvidia Titan V GPU. In our setup the data generation and network training are run efficiently through CPU parallelization in C# and GPU parallelization in Python respectively. However, transferring the CRP data between C# and Python creates a bottleneck that limits the amount of data we can load in a feasible amount of time to 2,000,000 CRPs. Re-coding of the neural network in C# using Microsoft's Cognitive Toolkit is needed to go beyond 2,000,000 CRPs.

For our neural network implementation please refer to our open source code publicly available on Github. Using the neural network configuration and training data described above, we are unable to model $\geq$ 8-XOR APUF with 64 bits, $\geq$ 7-XOR APUF with 128 bits and $\geq$ 6-XOR APUF with 256 bits given 2,000,000 CRPs. For the full neural network attack results please see Table 5.

8.3 Enhanced Reliability Based CMA-ES Attack On APUF

When modeling both APUF and XOR APUF designs, the original reliability based CMA-ES attack can be improved by using more precise fitness functions (see Section 4.2). This is significant because due to the improved modeling offered by $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$ and $(R_{\mathsf{absolute}}, R'_{\mathsf{absolute}})$, the reliability based CMA-ES attack can now work with less training data, where before the original attack would fail. When sufficient training data is available, the proposed fitness functions give more accurate models than the original fitness function. Table 6 depicts the modeling accuracy of the reliability based CMA-ES attack on a 64-bit APUF. In this set of simulations $\sigma_{\text{noise}} = \sigma/10$, giving the APUF a reliability of around $96 - 97\%$. From Table 6, it is evident that the reliability based CMA-ES attack using our proposed model $(R_{\mathsf{absolute}}, R'_{\mathsf{absolute}})$ and $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$ outperforms the reliability based CMA-ES attack using the original model $(R_{\mathsf{original}}, R'_{\mathsf{original}})$ as proposed in [Bec15]. This is due to both proposed models using the proper reliability ranges. In addition, $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$ outperforms $(R_{\mathsf{absolute}}, R'_{\mathsf{absolute}})$ as both the response polarity and reliability information are considered in $(R_{\mathsf{cdf}}, R'_{\mathsf{cdf}})$.

Table 6: A comparison of 64-bit APUF's modeling accuracy using CMA-ES

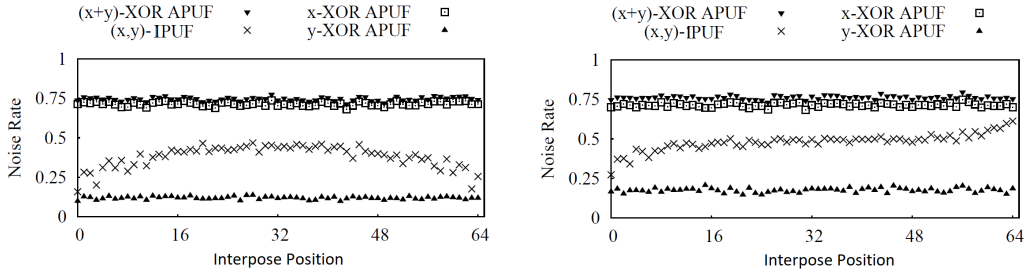
$N^\dagger$	**Modeling Accuracy(%)**		
	$(R_{\text{original}}, R'_{\text{original}})$	$(R_{\text{absolute}}, R'_{\text{absolute}})$	$(R_{\text{cdf}}, R'_{\text{cdf}})$
600	60.33	78.27	96.02
1500	69.60	96.64	97.80
3000	97.49	97.65	98.38
6000	97.85	97.98	98.40

† No. of CRPs is used to train a model.

Table 7: Modeling results of 4-XOR APUF using CMA-ES and different reliability models

Model setup	$N^\dagger$	**Modeling Acc.(%)**				**Frequency***			
		A_0	A_1	A_2	A_3	A_0	A_1	A_2	A_3
$(R_{o.}, R'_{o.})$	10×10^3	65.10	66.17	67.40	68.96	0	0	0	0
	20×10^3	98.08	97.91	98.23	98.33	1	4	3	1
	30×10^3	98.27	98.06	98.27	98.39	19	10	6	3
	50×10^3	98.31	98.16	98.37	98.44	39	20	17	10
$(R_{a.}, R'_{a.})$	10×10^3	97.53	97.09	97.10	95.24	22	24	31	8
	20×10^3	97.86	97.75	97.74	97.78	24	29	29	17
	30×10^3	98.08	97.89	98.02	98.12	47	27	20	6
	50×10^3	98.17	98.06	98.23	98.27	50	29	16	5

† No. of CRPs is used to train an APUF as well as 4-XOR APUF models.
⋆ No. of correct models (prediction accuracy > 90%) for A_i out of 100 runs of CMA-ES.



(a) $x = 20$ and $y = 2$ (b) $x = 20$ and $y = 3$

Figure 4: Noise rate of (x+y)-XOR APUF, (x,y)-iPUF, x-XOR APUF and y-XOR APUF when the noise rate of each APUF is around 0.05.

8.4 Enhanced Reliability Based CMA-ES Attack On XOR APUF

To demonstrate the improvement $(R_{\text{absolute}}, R'_{\text{absolute}})$ offers, we simulated a 4-XOR APUF and ran the reliability based CMA-ES attack 100 times using both $(R_{\text{absolute}}, R'_{\text{absolute}})$ and $(R_{\text{original}}, R'_{\text{original}})$. In this simulation each APUF has a $\sigma_{\text{noise}} = \sigma/10$, giving it a reliability of around $96 - 97\%$. The results of this simulation are shown in Table 7. We report two aspects in Table 7: i) modeling accuracy and ii) frequency of the correct APUF models (a correct model has a prediction accuracy greater than 90%). From the results of Table 7 it is clear that $(R_{\text{absolute}}, R'_{\text{absolute}})$ is able to generate more correct APUF models than $(R_{\text{original}}, R'_{\text{original}})$. Note that in the case where $N = 10 \times 10^3$, CMA-ES using $(R_{\text{absolute}}, R'_{\text{absolute}})$ can successfully model all the APUF components but $(R_{\text{original}}, R'_{\text{original}})$ fails to build any correct APUF models. Our enhancement to the CMA-ES attack results in a more efficient modeling of an XOR APUF.

8.5 Reliability Simulation of the iPUF and XOR APUF

To compare the reliability of the (x, y)-iPUF to the $(x + y)$-XOR APUF we simulated a $(x + y)$-XOR APUF, (x, y)-iPUF, x-XOR APUF and a y-XOR APUF for $x = 20$, and $y = 2$ and $y = 3$. In the simulation, we have 64-bit APUFs, each with a noise rate defined by setting $\sigma_{\text{noise}} = 0.05\sigma$. To estimate the reliability, we evaluated each PUF design with 10,000 randomly generated challenges. Each challenge is measured 11 times to determine whether it is noisy or not. If the repeatability of a challenge is 100%, we say it is reliable; otherwise, it is a noisy challenge. The reliability of a PUF is estimated as the fraction of reliable challenges.

We varied the interpose position i of the iPUF from 0 to 64. For each interpose position we measured the reliability of the (x, y)-iPUF. In the same figure we also plot the reliability of the $(x + y)$-XOR APUF, x-XOR APUF and y-XOR APUF. The simulated results are presented in Fig. 4.

Figures 4a and 4b show that the noise rate of the $(x + y)$-XOR APUF and the x-XOR APUF are close to each other as the value of y is very small compared to the value of x (i.e. $y = 2$ or $y = 3$). The noise rate of the y-XOR APUF is very small compared to the $(x + y)$-XOR APUF and the x-XOR APUF. The noise rate of the $(x, 3)$-iPUF increases when the interpose bit position increases from 0 to 64. However, the noise rate of the $(x, 2)$-iPUF reaches the maximum value at interpose position 32. This is due to the parity of y (see Eq. (17)). The noise rate of the $(x, 3)$-iPUF is equal to half of the noise rate of the $(x + 3)$-XOR APUF when i is in the middle position. The experiments confirm our findings related to reliability in Section 7.

9 iPUF Implementation

In order to validate our iPUF security and reliability claims, we implemented our proposed iPUF designs on an FPGA board. In this section, we describe the FPGA implementation details and related experimental results. We also discuss the limitations of FPGA based APUFs on composite PUF (XOR APUF and iPUF) security.

Good Implementation and Bad Implementation. Having a theoretical secure design is not enough. Besides a secure design, one also needs a proper implementation that meets the assumptions used in the security analysis of the design. The *hidden* assumption used in the security analysis of the iPUF is that we have *good* APUF instances, i.e., all the weights $\mathbf{w}[i]$ (see Eq. 1) of the APUFs are sampled from the same distribution $\mathcal{N}(0, \sigma^2)$. Failure of this condition in a practical APUF implementation would reduce or severely damage the security of the iPUF. In this section, we will show that badly implemented APUFs lead to a vulnerable iPUF which can be broken by the reliability-based CMA-ES attack. We show that a properly implemented iPUF defeats the reliability-based CMA-ES attack.

9.1 iPUF Implementation Details

In our iPUF design we implemented every stage (switch) of each APUF by using a Look-Up Table (LUT)[7] on a Digilent Nexys 4 DDR board with Xilinx Artix-7 embedded [Dig16]. The LUTs are chained together and the final output is collected by a flip-flop serving as an arbiter. The main issue in implementing any iPUF design is to make sure that the

[7] In particular on our board, each stage is implemented by a 6-input 2-output LUT, where the two outputs serve as the outputs of upper and lower paths and only three inputs are used as the inputs of upper and lower paths and the challenge bit. We notice that Programmable Delay Line (PDL) is a widely used technique to implement APUFs [MKD10], but we did not choose to use it due to the poor uniqueness of PDL-based APUFs according to a recent study [SCM15].

response r_{up} of the upper layer XOR APUF is ready when the interposed position of the lower layer XOR APUF is evaluated. To solve this issue, we added one more signal from the upper layer XOR APUF to inform the control circuitry when r_{up} is ready. Once the signal is received the lower layer XOR APUF evaluates its input.

Desirable statistical properties of a PUF include uniqueness, reliability and uniformity. It has been experimentally verified that uniqueness is a serious issue when implementing FPGA based APUFs [MGS11, HYKS10]. The main reason implementing unique APUFs on FPGAs is problematic [MYIS14, MGS11] is because the designers are not allowed to precisely control the routing between each LUT. The delay difference induced by routing is much larger than the delay difference introduced by process variation. Thus, the behavior of one APUF on an FPGA is largely determined by the placement and routing of the LUTs. Controlling the routing between the LUTs maintains the balance of the length of the two competing paths.

In practice, in order to keep the balance between the delays of two competing paths, usually one switch chain is kept in one column of slices on an FPGA. However, in our APUF implementation on a Digilent Nexys 4 DDR with Xilinx Artix-7 embedded [Dig16] there are four LUTs in each slice of the FPGA. Due to the configuration on this hardware, a choice must be made on how to connect the LUTs inside a slice or between two adjacent slices to form the switch chain. Since the behavior of each APUF is dominated by the routing difference between the two paths, each switch chain must be placed differently for each APUF, in order to create unique APUFs. Note this design strategy only alleviates the uniqueness issue on a single FPGA. If the same bitstream is used to program different FPGAs, the difference between the same APUFs on different FPGAs will be very small. Thus, this is not a general design strategy to improve the uniqueness of APUFs on FPGA, but it is sufficient for us to conduct our experiments.

We have two different ways to place the switch chains: (1) Random placement [**Bad Implementation**]: we randomly select one LUT in each slice and then connect them together. (2) Pattern placement [**Good Implementation**]: we place the switches according to a pre-defined pattern. For example, we only use LUT A and LUT B in every slice.

According to our experiments, each design strategy has advantages and disadvantages. For the random placement design strategy, the advantage of this method is that it gives us many options to build unique APUFs. Here we define two APUFs as being non-unique when their responses are the same for more than 60% of the challenges[8]. Initially, we generated 100 different switch chains using random placement. After extensive evaluation, we selected 23 placements of the APUFs, which can provide APUFs with good uniformity (50.2% - 61.6%) and good uniqueness/inter-hamming distance (39.5% - 59.0%). The noise rate of these APUFs under room temperature is between 0.66% and 1.25%. However, with good statistic properties, comes a security weakness in the APUFs. Since the routing between each adjacent switches is randomly selected, there are a few delays that are significantly larger than the other delays. This effectively introduces a few significant weights in the weight vector (see Eq. 1) of this APUF. As a result the difficulty of all machine learning attacks is reduced since only these significant weights need to be precisely modeled (instead of having to precisely learn all the weights). *This gives a bad PUF implementation and therefore it can now be broken by ML attacks.* We built the model of individual APUFs to understand the distribution of weights. The standard deviation of the weights of the APUFs created by random placement ranges from 4.23 to 7.48. As we will explain shortly, the standard deviation is much smaller for the pattern placement design strategy, but this method has its own drawbacks.

For the pattern placement design strategy, there are a very limited number of patterns we can generate and some APUFs formed by different placement patterns are not unique.

[8]For FPGA implemented APUFs, 60% uniqueness is considered sufficiently close on its ideal value 50%, due to the difficulty of controlling the routing between LUTs. As it is shown in our experiments, only 23 out of 100 can satisfy this requirement. Our results are in agreement with [SCM15].

Table 8: Results of reliability based machine learning attack on XOR APUFs with real measurements from the FPGA.

	#CRPs used in one attack	Overall Noise Rate	Average Prediction Accuracy
2-XOR	50,000	1.44%	98.22%
3-XOR	90,000	2.38%	96.38%
4-XOR	140,000	2.92%	96.15%
5-XOR	200,000	3.80%	96.62%
6-XOR	260,000	4.47%	91.58%*

*Note that the attack on 6-XOR APUF only recovered 5 out of 6 models. The attacks on 2,3,4,5 XOR APUFs successfully recovered all the models.

After exhaustively trying 11 patterns[9], we found only 4 placement patterns that can generate 4 unique APUFs. The uniqueness limitation of this design strategy gives us very few options to form an iPUF design with more than 4 APUFs in total. However, the pattern placement design strategy does not have the same security weakness as the random placement design strategy. *This is a good PUF and the PUF is secure against ML attacks.* We also built the model of individual APUFs to understand the distribution of weights in this design. The standard deviation of the weights of the APUFs created by pattern placement ranges from 1.11 to 3.77. Using this design strategy we do not observe the same effect of only a few influential weights (unlike in the random placement design strategy).

Since the design strategy will largely influence the experimental results that we will present later, we will clearly state how the APUFs are generated for each experiment.

9.2 Experimental Results

Reliability-Based Machine Learning Attack on XOR APUFs: First, we repeated the enhanced reliability based CMA-ES attack in Section 8.4 on XOR APUFs to validate the effectiveness of our attack. In this experiment, we selected 6 unique APUFs created by random placement to form a 6-XOR APUF on the FPGA. We then measured 300,000 CRPs with each CRP measurement repeated 11 times to get 300,000 challenge reliability pairs. The number of challenge reliability pairs used for one CMA-ES attack and the modeling results after 100 runs of CMA-ES are presented in Table 8.

Reliability-Based Machine Learning Attack on iPUFs: To show the machine learning resistance of the iPUF to the enhanced reliability-based CMA-ES attack, we perform the attack on a (1,1)-iPUF. We do not test this attack on iPUF designs with more APUF components because our security against reliability-based machine learning attacks does not depend on the number of APUF components.

We used two APUFs created by pattern placement [**Good Implementation**] to form a (1,1)-iPUF with an interposed position exactly in the middle in the lower APUF. We measured 200,000 CRPs with each CRP measurement repeated 11 times to get the challenge reliability measurements. We then sampled a subset of 90,000 challenge reliability pairs out of the 200,000 challenge reliability pairs to run the reliability-based CMA-ES attack. CMA-ES was not able to converge to the upper APUF after 10 runs. This validated our security claim that the iPUF structure can prevent reliability based machine learning attacks. Unfortunately, due to the uniqueness issue of pattern placed APUFs, we were not able to test the security of the (3,3)-iPUF as we did in our experimental simulations. We also observed that if the component APUFs are created by random placement [**Bad Implementation**], CMA-ES was able to break it. We give a detailed analysis of this

[9]Since there are only four LUTs (A, B, C, and D) in a slice, we can only construct 11 different patterns: 1 pattern that uses all four LUTs in each slices, 6 patterns that uses a combination of two LUTs in each slices (AB, AC, AD, BC, BD, and CD), 4 patterns that uses one LUTs in each slices (A, B, C, and D).

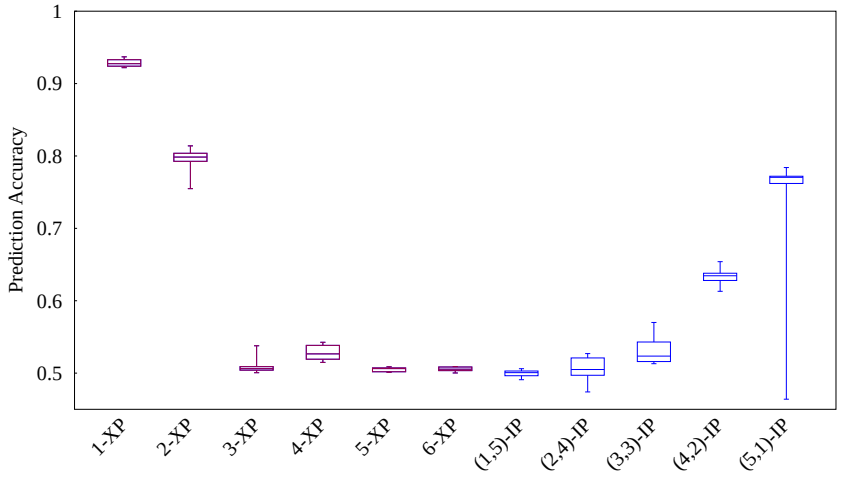


Figure 5: Results of CRP based CMA-ES attacks on APUF (1-XP), x-XOR APUFs (x-XP) and $((x, y))$-iPUFs $((x, y)$-IP).

phenomenon in Section 9.3. We want to clarify the fact that the reliability-based CMA-ES attack can break the iPUF because of the **improper implementation in case of random placement**, i.e., **not because of any design flaws of the iPUF**.

Classical Machine Learning Attacks on XOR APUFs and iPUFs: In this subsection we conduct experiments to verify the claim that the model complexity of an (x, y)-iPUF is similar to that of a $(\mathrm{y}+\frac{x}{2})$-XOR APUF. In this experiment we generate component APUFs using the random placement design strategy to construct iPUFs and XOR APUFs.

We measured 200,000 CRPs for each XOR APUF and iPUF. We then used CRP based CMA-ES to optimize each model given the 200,000 CRPs. To further reduce the influence of noisy CRPs in this attack, for each CRP we did a majority voting from 11 repeated measurements. We ran CRP based CMA-ES on this training dataset 10 times to avoid the possible failure introduced by the probabilistic nature of the algorithm. The results are shown in Fig. 5

Strict Avalanche Property: We also tested SAC property of an implemented (3,3)-iPUF, where each component APUF is generated by random placement (see Section 9.1). The shape is shown in Figure 6, which is similar to Figure 3.

Reliability of the iPUF with respect to Interposed Position: We selected 22 and 23 unique component APUFs to construct a (20,2)-iPUF and a (20,3)-iPUF respectively on the FPGA. We evaluated how the noise rate was affected by changing the interposed positions in the lower XOR APUFs. We tested 5 different interposed bit positions (0, 16, 32, 48, 64). The noise rate of the (20,2)-iPUF and the (20,3)-iPUF with respect to interposed position are shown in Fig. 7. This follows the same trend as the simulation result in Fig. 4. Note that the reliability is equal to (1-noise rate) or (100% − noise rate) in percentage.

Reliability of iPUF under Temperature Variation: We used a $(3, 3)$-iPUF for reliability testing under temperature variation, where each APUF is created by random placement, and has good uniformity and uniqueness. We measured 1,000 CRPs from this iPUF under 25 degree Celsius as the reference CRPs. We then measured the same 1,000 CRPs again under 70 degrees and 0 degrees Celsius. The error rate introduced by 70

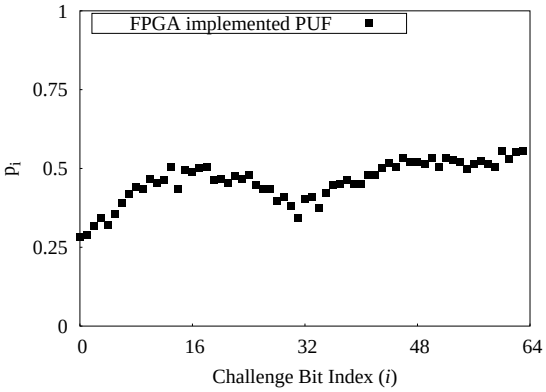


Figure 6: SAC property of the 64-bit (3,3)-iPUF with real data.

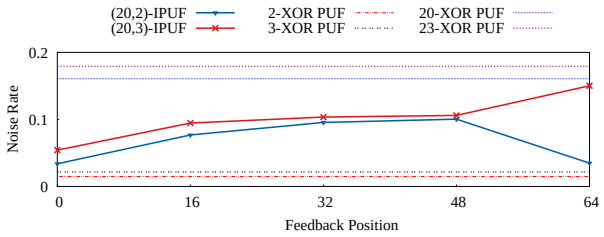


Figure 7: Reliability with respect to different interpose position.

degrees and 0 degrees Celsius is 2.1% and 1.4%, respectively.

9.3 Security Issue Introduced by Careless Implementation

We also used two APUFs created by random placement to form a (1,1)-iPUF with an interposed position exactly in the middle in the lower APUF. Again, we measured 200,000 CRPs with each CRP measurement repeated 11 times to get the challenge reliability measurements. We then sampled a subset of 90,000 challenge reliability pairs out of the 200,000 challenge reliability pairs to run the reliability-based CMA-ES attack. CMA-ES was able to converge to the model of the upper APUF with 96.6% accuracy. According to our previous analysis and simulation results, only the model of the lower APUF should be built with up to 75% accuracy (i.e., the reliability-based machine learning attack on a (1,1)-iPUF is equivalent to the linear approximation reliability-based CMA-ES). When CMA-ES converged to the model of the lower APUF, the accuracy was upper bounded by 75%, which confirms our theoretical analysis.

When we discovered the phenomenon of biased weights in the APUFs generated by random placement, we further investigated why this implementation affects the security of iPUFs. We created APUF models with biased weights, such that the distribution that generates the large weights is $\mathcal{N}(0, 6)$, while the standard weights are drawn from $\mathcal{N}(0, 1)$. We also precisely controlled the number of large weights in the second half (the half which is closer to the output) of the lower APUF. We simulated (1,1)-iPUFs with 4, 8, 12, 16, and 20 dominating weights. On each type of iPUF we performed the enhanced reliability-based CMA-ES attack 10 times. The results are shown in Table 9.

From Table 9, we can see that the SAC value of the middle bit of the lower APUF will affect its security. The SAC value is computed by the probability that a bit flip in the middle challenge bit of the lower APUF will flip the final output bit. The larger the SAC value is, the higher the chance that the reliability information of the upper APUF will be

Table 9: Results of the enhanced reliability-based CMA-ES attack on a (1,1)-iPUF with biased weights.

	SAC	Upper	Lower	Failed
4	0.39	0	10	0
8	0.58	9	1	0
12	0.54	2	8	0
16	0.32	0	9	1
20	0.41	0	10	0

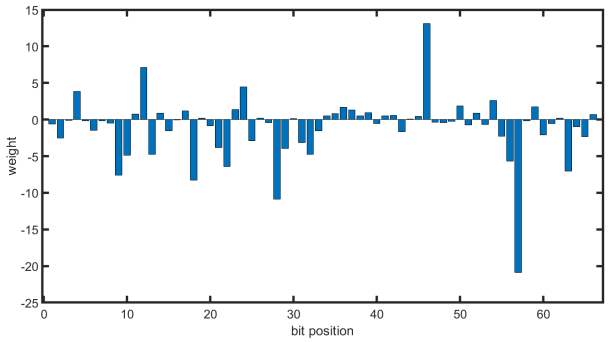


Figure 8: The weight distribution of the lower APUF in a (1,1)-iPUF where the number of large weights (> 3) is constrained to 4.

exposed to adversaries. This leads to a successful attack on the upper APUF with higher probability. Ideally, the SAC value of the middle bit should be 50%, but due to the biased distribution in the weights on FPGAs, the SAC value can be possibly higher than normal. From the computed SAC values, the reliability-based CMA-ES attack **is supposed to not** converge to the **upper** APUF because they are good enough to prevent the attack according to our analysis in Section 6.4. However, the attack still works very well when the number of dominating weights is 8 or 12.

Security Analysis. We can explain this fact as follows. The interposed bit at the middle splits the lower APUF into two parts: the flipping part which is closer to the 0-th challenge bit and the non-flipping part which is closer to the output of iPUF. For the sake of explanation, we assume that all small weights are equal to 0. The motivation of assuming this is that if the biased weights are significant larger than the small weights, we can ignore all the small weights. We assume that there are k biased weights in the flipping part and m biased weights in the non-flipping part. Moreover, the biased weights follow a normal distribution $\mathcal{N}(0, \sigma_b^2)$. From Equation (13), we can write

$$\Delta = (1 - 2\mathbf{c}[n/2]) \times \Delta_{Flipping} + \Delta_{Non-Flipping},$$

where $\Delta_{Flipping} = \sum_{i=0}^{n/2} \mathbf{w}[i] \frac{\mathbf{\Psi}[i]}{(1-2\mathbf{c}[n/2])}$ and $\Delta_{Non-Flipping} = \sum_{i=n/2+1}^{n} \mathbf{w}[i]\mathbf{\Psi}[i]$. Since we have $k + m$ biased weights, $\Delta_{Flipping}$ and $\Delta_{Non-Flipping}$ only depend on k and m weights, respectively. This implies $\Delta_{Flipping}$ and $\Delta_{Non-Flipping}$ follow normal distributions $\mathcal{N}(0, k\sigma_b^2)$ and $\mathcal{N}(0, m\sigma_b^2)$, respectively. We know that, for a given challenge $\mathbf{c}$, the output of the lower APUF would flip by flipping challenge bit $\mathbf{c}[n/2]$ when $|\Delta_{Non-Flipping}| < |\Delta_{Flipping}|$. It is obvious that the smaller $|\Delta_{Non-Flipping}|$ (let's say $< e$), the higher the chance the output will be flipped. Since $\Delta_{Non-Flipping} \sim \mathcal{N}(0, m\sigma_b^2)$,

$$\begin{aligned}\Pr(|\Delta_{Non-Flipping}| < e) &= \Phi(\frac{e}{\sqrt{m}\sigma_b}) - \Phi(\frac{-e}{\sqrt{m}\sigma_b}) \\ &= 2\Phi(\frac{e}{\sqrt{m}\sigma_b}) - 1.\end{aligned}$$

If m increases, then $\Pr(|\Delta_{Non-Flipping}| < e)$ decreases for any given small positive number e. This implies that the leakage of information is reduced when increasing m. This explains why the number of convergences to the upper APUF for the case $m = 8$ is larger than for when $m = 12$. Of course, when m is large enough, the attack does not work. In our simulation, we just need m to be larger than 16 to prevent the attack. In our simulation, we noticed that when $m = 4$, the attack does not work. The weights of lower APUF when $m = 4$ is described in Fig. 8. We give the following possible reasoning. In the non-flipping part, there are two significant large weights and the second largest weight among the two is much smaller (< 13) than the largest one (≈ 23). Moreover, both of them are much bigger than the remaining weights. The smallest value of $|\Delta_{Non-Flipping}|$ is 10 and largest value is 36. This implies that $|\Delta_{Non-Flipping}|$ is always larger than $|\Delta_{Flipping}|$. It means that the leakage of the output of the upper APUF is very small and thus, the attack does not work. Actually in this case, the SAC property for $m = 4$ is 0.39 which is smaller than ideal SAC of 0.5.

Discussion. *What if the biased delay value is at the interposed stages of the lower y-XOR APUFs in an iPUF?*

We will discuss this situation in two cases: (a) Suppose that the delay values at the interposed stages of the lower y-XOR APUFs are very small comparing with the other delay values, then the delay introduced by the interposed stages are very small, but the response of the upper x-XOR APUF still affects the final response of y-XOR APUF. This is because the challenge bit decides whether to swap the accumulated delay values of all the stages in front of the interposed stage, and this still introduces a large impact to the final response of y-XOR APUF and the iPUF as a whole. (b) The second case is if the interposed stages have a very large delay that is comparable with the accumulated delay of all the other stages in the y-XOR APUF. In this case then the final response of the iPUF will be largely influenced by the x-XOR APUF and it will be defeated by the reliability-based CMA-ES attack.

To comment on the possibility of this event, we note that it is extremely rare that the delay values at the interposed stages of every single APUF in the y-XOR APUF are larger than the accumulated delays of all the other stages in a good implementation. Thus, this will not be a concern in terms of security for a correctly implemented iPUF.

10 Conclusion

In this paper, we develop three main contributions. First, we comprehensively analyzed the reliability-based CMA-ES attack to understand how it works and how to enhance it. Second we propose a new PUF design, the (x, y)-iPUF. We prove through theory and experimentation that the iPUF is not vulnerable to the strongest known reliability-based machine learning attack (reliability-based CMA-ES) and the strongest known classical machine learning attack (Logistic Regression). Our final contribution is publicly available source code for all our iPUF, XOR APUF and APUF attack simulations written in Matlab and C#. We also provide source code for the FPGA implementation of the XOR APUF and iPUF. All codes for this paper can be found in DA PUF Library on Github. Since the iPUF has more advantages in terms of security, reliability and hardware overhead compared to the XOR APUF, **it implies that the iPUF can be considered a standard design or primitive replacement for the XOR APUF**. As a next step, we propose to implement the iPUF in ASIC (Application Specific Integrated Circuit) in order to overcome the uniqueness problem encountered in FPGA implementations.

A Appendix

A.1 The Arbiter PUF and its XORed Versions

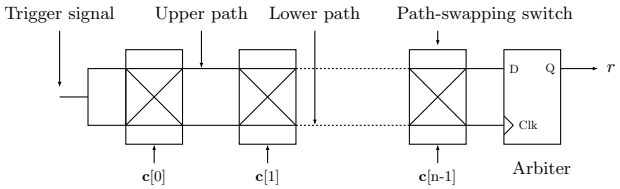


Figure 9: Arbiter PUF.

We briefly introduce the design of Arbiter PUFs (APUFs) and the reader is referred to [GCvDD02] for further details. The design of an APUF is depicted in Fig. 9. It is a delay-based silicon PUF with n-bit challenge $\mathbf{c} = (\mathbf{c}[0], \mathbf{c}[1], \ldots, \mathbf{c}[n-1])$ that extracts random variation in silicon in terms of the delay difference of two symmetrically laid out parallel delay lines. Ideally, the nominal delay difference between these path pairs should be 0, but this does not happen due to uncontrollable random variation in the manufacturing process that introduces random offset between the two path delays. In general, an n-bit APUF comprises of n switches connected serially to build two distinct, but symmetrical paths. The arbiter which is located at the end of the two paths is used to decide which path is faster. The challenge bits $\mathbf{c}[0], \mathbf{c}[1], \ldots, \mathbf{c}[n-1]$ are used as the control input of path-swapping switches that eventually results in two paths and input stimulus runs through these two paths. The arbiter declares which path wins the race in the form of a 0 or 1 response. Typically, the response of an APUF is defined by

$$r = \begin{cases} 1, & \text{if the signal at the upper path runs faster} \\ 0, & \text{otherwise.} \end{cases}$$

The most important feature of this design is its small hardware overhead, i.e. the hardware overhead of an n-bit APUF is linearly proportional to the number of challenge bits n.

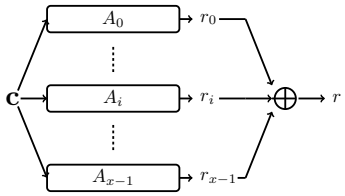


Figure 10: x-XOR APUF.

Due to the existence of a linear additive delay model of the APUF, see Eq.(1), a modeling attack is applicable [LLG+05, RSS+10]. In [SD07], the authors proposed the design of an x-XOR APUF which enjoys better modeling resistance. Figure 10 describes the design of an x-XOR APUF.

A.2 Analysis of Influence of Challenge Bit $\mathrm{c}[j]$ on the Response r in the APUF

In this section, we explain

$$\Pr_{\mathbf{c}}(r_{\mathbf{c}[j]=0} = r_{\mathbf{c}[j]=1}) \approx \frac{(n-j)}{n}, j = 0, \ldots, n-1.$$

Let us examine under which conditions $r_{\mathbf{c}[j]=0}$ will equal $r_{\mathbf{c}[j]=1}$. Assume for a specific challenge $\mathbf{c}$ we fix all bits (except for $\mathbf{c}[j]$) and the output r is 0 regardless of the value of $\mathbf{c}[j]$. This means $\Delta_{\mathbf{c}[j]=0} = \Delta_{Flipping} + \Delta_{Non-Flipping} > 0$ when $\mathbf{c}[j] = 0$ and $\Delta_{\mathbf{c}[j]=1} = -\Delta_{Flipping} + \Delta_{Non-Flipping} > 0$ when $\mathbf{c}[j] = 1$. From this example, it is easy to derive that $r_{\mathbf{c}[j]=0} = r_{\mathbf{c}[j]=1}$ if and only if $|\Delta_{Flipping}| \leq |\Delta_{Non-Flipping}|$ so that:

$$\Pr_{\mathbf{c}}(r_{\mathbf{c}[j]=0} = r_{\mathbf{c}[j]=1}) = \Pr(|\Delta_{Flipping}| \leq |\Delta_{Non-Flipping}|). \tag{29}$$

We follow existing PUF literature [Lim04, LP14] and assume that when generating an instance of an APUF all $\mathbf{w}_i$ are sampled from the same normal distribution $\mathcal{N}(0, \sigma^2)$ and hence, $\Delta_{Flipping} \sim \mathcal{N}(0, (j+1) \times \sigma^2)$ and $\Delta_{Non-Flipping} \sim \mathcal{N}(0, (n-j) \times \sigma^2)$. Thus $\Pr_{\mathbf{c}}(r_{\mathbf{c}[j]=0} = r_{\mathbf{c}[j]=1})$ is equal to:

$$\begin{aligned} &= \Pr(|\Delta_{Flipping}| \leq |\Delta_{Non-Flipping}|) \\ &= 4 \times \int_0^\infty \phi_{0,(n-j)\sigma^2}(u) \Phi_{0,(j+1)\sigma^2}(-u)\, \mathrm{d}u, \end{aligned} \tag{30}$$

where $\phi_{\mu,\sigma^2}(\cdot)$ and Φ_{μ,σ^2} are the probability distribution function and cumulative distribution function of a normal distribution $\mathcal{N}(\mu, \sigma^2)$, respectively. Experimentally it has been shown [MKP09, DGSV14, NSCM16] that Eq. (30) can be approximated as:

$$\Pr_{\mathbf{c}}(r_{\mathbf{c}[j]=0} = r_{\mathbf{c}[j]=1}) \approx \frac{(n-j)}{n}, j = 0, \ldots, n-1.$$

A.3 MPUF's Vulnerability

MPUF was introduced in [SMCN18] but it is also not secure against the reliability-based CMA-ES attack (see Algorithm 1 in Section 5 in [SMCN18]). To make the MPUF more robust against the reliability-based CMA-ES attack, the authors propose rMPUF (see Figure 3 in [SMCN18]). The rMPUF can be broken by using a modified version of Algorithm 1 and the reliability-based CMA-ES attack as well. Note that the notation A_s is proposed in Figure 3 in [SMCN18]. First we learn the APUF A_s on the multiplexer closest to the output using the reliability-based CMA-ES attack (this PUF has more noisy CRPs in the dataset, so the reliability-based CMA-ES attack is more likely to converge to this model). Next we use our learned model to determine which CRPs are reliable for A_s. Using these A_s reliable CRPs we create a new dataset for the rMPUF and run the reliability-based CMA-ES attack on the new dataset to learn the next PUF closest to the output (since the new dataset only has CRPs that are reliable for A_s we know that the reliability-based CMA-ES attack will most likely converge to the next nosiest PUF). We can repeat this process until we have learned all the APUFs in the rMPUF. It is important to note that this attack works on the rMPUF, this attack does not work on the iPUF because we cannot learn any of the APUFs in the iPUF. Clearly, this approach cannot be applied to iPUF because it requires the reliability-based CMA-ES attack to learn at least one APUF component in every step.

A.4 $\tilde{k}$-junta Test

To test the number of influential inputs in a given function, $\tilde{k}$-junta test is usually applied to the function. The concept of $\tilde{k}$-junta is defined as follows:

Definition 3. [**$\tilde{k}$-junta**] A function $f : \Omega^n \to Z$ is a $\tilde{k}$-junta iff there is a subset $J \subseteq [n]$ with $|J| \leq \tilde{k}$ and function $g : \Omega^J \to Z$ such that for all $x \in \Omega^n$, $f(x) = g(x_J)$ where $x_J = (x_{j1}, ..., x_{j\tilde{k}})$ for $J = \{j_1, ..., j_{\tilde{k}}\}$.

The $\tilde{k}$-junta test [Bea] algorithm can help us determine whether a given function f

- f is a $\tilde{k}$-junta, or
- f is ϵ-far from any $\tilde{k}$-junta, where we say two functions f and g are ϵ-far under a certain distribution μ iff $P_{x\sim\mu}[f(x) \neq g(x)] > \epsilon$.

The algorithm of $\tilde{k}$-junta test works as follows [Bea]:

Algorithm 1 $\tilde{k}$-junta test [Bea]

procedure JUNTATEST(f, $\tilde{k}$, ϵ)
  $s \leftarrow (\tilde{k}/\epsilon)^{O(1)}$
  $t \leftarrow 12(\tilde{k}+1)/\epsilon$
  Randomly partition $[n]$ in s sets $I_1, ..., I_s$; i.e., choose a random $h : [n] \rightarrow [s]$ and let $I_j = \{i|h(i) = j\}$.
  $S \leftarrow [n]$
  $l \leftarrow 0$
  for $r \leftarrow 1$ to t **do**
    Choose x, y independently from Ω^n
    $y_{\bar{s}} \leftarrow x_{\bar{s}}$
    if $f(x) \neq f(y)$ **then**
      Use binary search on $[s]$ to find a set I_j such that I_j contains an influential variable
      Remove I_j from S
      $l \leftarrow l + 1$
      if $l > \tilde{k}$ **then**
        Output "Not a $\tilde{k}$-junta" and halt
      end if
    end if
  end for
  Output "$\tilde{k}$-junta"
end procedure

Since it only generates output with probability $> \frac{2}{3}$, in practice we have to run this test repeatedly to gain a higher confidence in the result.

References

[Bea] Paul Beame. Lecture 16: Property Testing of Functions, Junta Testing. Technical report.

[Bec14] Georg T. Becker. On the Pitfalls of using Arbiter-PUFs as Building Blocks. *IACR Cryptology ePrint Archive*, 2014:532, 2014.

[Bec15] Georg T. Becker. The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs. In *Proc. of 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2015.

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schrauder, and Stefan Katzenbeisser. Physically Uncloneable Functions in the Universal Composition Framework. In Phillip Rogaway, editor, *CRYPTO*, pages 51–70. 2011.

[CCL+11] Qingqing Chen, G. Csaba, P. Lugli, U. Schlichtmann, and U. R ührmair. The Bistable Ring PUF: A new architecture for strong Physical Unclonable Functions. In *Proc. of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 134 –141, june 2011.

[DGSV14] Jeroen Delvaux, Dawu Gu, Dries Schellekens, and Ingrid Verbauwhede. Secure Lightweight Entity Authentication with Strong PUFs: Mission Impossible? In *Proc. of 16th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 451–475, 2014.

[Dig16] Digilent. Nexys 4 DDR Reference Manual, Apr. 2016. [Accessed Feb., 2018].

[DR02] Joan Daemen and Vincent Rijmen. *The Desigsn of Rijndael*. Springer-Verlag, Berlin, Heidelberg, 2002.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In Christian Cachin and Jan L. Camenisch, editor, *EUROCRYPT*, pages 523–540. 2004.

[DV13] J. Delvaux and I. Verbauwhede. Side Channel Modeling Attacks on 65nm Arbiter PUFs Exploiting CMOS Device Noise. In *IEEE 6th Int. Symposium on Hardware-Oriented Security and Trust* , 2013.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[GCvDD02] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In *ACM CCS*, 2002.

[GTFS16] Fatemeh Ganji, Shahin Tajik, Fabian Fäßler, and Jean-Pierre Seifert. Strong Machine Learning Attack against PUFs with No Mathematical Model. In *CHES*, pages 391–411. Springer Berlin Heidelberg, 2016.

[GTS15] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. Why attackers win: on the learnability of XOR arbiter PUFs. In *International Conference on Trust and Trustworthy Computing*, pages 22–39. Springer International Publishing, 2015.

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

[Han16] Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. *CoRR*, abs/1604.00772, 2016.

[HAP09] Ryan Helinski, Dhruva Acharyya, and Jim Plusquellic. A physical unclonable function defined using power distribution system equivalent resistance variations. In *Proc. of 46th Annual Design Automation Conference(DAC)*, pages 676–681, New York, NY, USA, 2009. ACM.

[HRvD+17] Charles Herder, Ling Ren, Marten van Dijk, Meng-Day Yu, and Srinivas Devadas. Trapdoor computational fuzzy extractors and stateless cryptographically-secure physical unclonable functions. *IEEE Transactions on Dependable and Secure Computing*, 14(1):65–82, 2017.

[HYKS10] Y. Hori, T. Yoshida, T. Katashita, and A. Satoh. Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs. In *Proceedings of International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 298 –303, dec. 2010.

[JHR+17] Chenglu Jin, Charles Herder, Ling Ren, Phuong Ha Nguyen, Benjamin Fuller, Srinivas Devadas, and Marten van Dijk. FPGA Implementation of a Cryptographically-Secure PUF Based on Learning Parity with Noise. *Cryptography*, 1(3):23, 2017.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[KGM+08] S.S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls. Extended abstract: The butterfly PUF protecting IP on every FPGA. In *HOST*, pages 67–70, June 2008.

[Lim04] Daihyun Lim. Extracting Secret Keys from Integrated Circuits. Master's thesis, MIT, USA, 2004.

[LLG+05] Daihyun Lim, Jae W. Lee, Blaise Gassend, G. Edward Suh, Marten van Dijk, and Srini Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205, October 2005.

[LP14] Yingjie Lao and Keshab K. Parhi. Statistical Analysis of MUX-Based Physical Unclonable Functions. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 33(5):649–662, 2014.

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

[MKD10] Mehrdad Majzoobi, Farinaz Koushanfar, and Srinivas Devadas. Fpga puf using programmable delay lines. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pages 1–6. IEEE, 2010.

[MKP08] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Testing Techniques for Hardware Security. In *Proc. of IEEE International Test Conference(ITC)*, pages 1–10, Oct. 2008.

[MKP09] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Techniques for Design and Implementation of Secure Reconfigurable PUFs. *ACM Trans. Reconfigurable Technol. Syst.*, 2(1):1–33, 2009.

[MOV96] A. Menezes, P.V. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

[MV10] Roel Maes and Ingrid Verbauwhede. Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 3–37. Springer, Berlin Heidelberg, 2010.

[MYIS14] Takanori Machida, Dai Yamamoto, Mitsugu Iwamoto, and Kazuo Sakiyama. A New Mode of Operation for Arbiter PUF to Improve Uniqueness on FPGA. In *Proc. of Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 871–878, 2014.

[NSCM16] Phuong Ha Nguyen, Durga Prasad Sahoo, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. Security Analysis of Arbiter PUF and Its Lightweight Compositions Under Predictability Test. *ACM TODAES*, 22(2):20, 2016.

[RJB+11] U. Rührmair, C. Jaeger, M. Bator, M. Stutzmann, P. Lugli, and G. Csaba. Applications of High-Capacity Crossbar Memories in Cryptography. *IEEE Transactions on Nanotechnology*, 10(3):489 –498, may 2011.

[RSS+10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling attacks on physical unclonable functions. In *Proc. of 17th ACM conference on Computer and communications security(CCS)*, pages 237–249, New York, NY, USA, 2010. ACM.

[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 *Proc. of 16th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pages 476–492, 2014.

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

[Sch16] M. Georg. Schroder. Logistic Regression A Self Learning Text. 2016.

[SCM15] Durga Prasad Sahoo, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. Towards ideal arbiter puf design on xilinx fpga: A practitioner's perspective. In *Digital System Design (DSD), 2015 Euromicro Conference on*, pages 559–562. IEEE, 2015.

[SD07] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *DAC*, pages 9–14, 2007.

[SH14] Dieter Schuster and Robert Hesselbarth. Evaluation of Bistable Ring PUFs Using Single Layer Neural Networks. In *Trust and Trustworthy Computing - 7th International Conference, TRUST 2014, Heraklion, Crete, Greece, June 30 - July 2, 2014. Proceedings*, pages 101–109, 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 Transactions on Computers*, 67(3):403–417, 2018.

[SNMC15] Durga Prasad Sahoo, Phuong Ha Nguyen, Debdeep Mukhopadhyay, and Rajat Subhra Chakraborty. A Case of Lightweight PUF Constructions: Cryptanalysis and Machine Learning Attacks. *IEEE TCAD*, 2015.

[Söl09] Jan Sölter. Cryptanalysis of Electrical PUFs via Machine Learning Algorithms. Master's thesis, Technische Universität München, 2009.

[TB15] Johannes Tobisch and Georg T. Becker. On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation. In *Proc. of 11th International Workshop on Radio Frequency Identification: Security and Privacy Issues (RFIDsec)*, pages 17–31, 2015.

[TDF+14] Shahin Tajik, Enrico Dietz, Sven Frohmann, Jean-Pierre Seifert, Dmitry Nedospasov, Clemens Helfmeier, Christian Boit, and Helmar Dittrich. Physical Characterization of Arbiter PUFs. In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 493–509, 2014.

[TLG+15] S. Tajik, H. Lohrke, F. Ganji, J. P. Seifert, and C. Boit. Laser Fault Attack on Physically Unclonable Functions. In *12th Workshop on Fault Diagnosis and Tolerance in Cryptography (FTDC)*, 2015.

[WGM+17] Nils Wisiol, Christoph Graebnitz, Marian Margraf, Manuel Oswald, Tudor Soroceanu, and Benjamin Zengin. Why Attackers Lose: Design and Security Analysis of Arbitrarily Large XOR Arbiter PUFs. 2017.

[XRHB15] Xiaolin Xu, Ulrich Rührmair, Daniel E. Holcomb, and Wayne P. Burleson. Security Evaluation and Enhancement of Bistable Ring PUFs. In *Proc. of 11th International Workshop on Radio Frequency Identification: Security and Privacy Issues (RFIDsec)*, pages 3–16, 2015.

[YHD+16] Meng-Day Yu, Matthias Hiller, Jeroen Delvaux, Richard Sowell, Srinivas Devadas, and Ingrid Verbauwhede. A lockdown technique to prevent machine learning on PUFs for lightweight authentication. 2016.

[YKL+13] Yida Yao, Myung-Bo Kim, Jianmin Li, Igor L. Markov, and Farinaz Koushanfar. ClockPUF: Physical Unclonable Functions based on Clock Networks. In *Design, Automation & Test in Europe (DATE)*, Grenoble, France, 2013.

[YMSD11] Meng-Day (Mandel) Yu, David M'Raïhi, Richard Sowell, and Srinivas Devadas. Lightweight and Secure PUF Key Storage Using Limits of Machine Learning. In *CHES*, pages 358–373, 2011.