Systematically Quantifying Cryptanalytic Non-Linearities in Strong PUFs

Durba Chatterjee*, Kuheli Pratihar*, Aritra Hazra*, Ulrich Rührmair†, Debdeep Mukhopadhyay*
*Indian Institute of Technology Kharagpur † LMU München, München, Germany, and University of Connecticut, Storrs, USA

***Abstract*—Physically Unclonable Functions (PUFs) with large challenge space (also called Strong PUFs) are promoted for usage in authentications and various other cryptographic and security applications. In order to qualify for these cryptographic applications, the Boolean functions realized by PUFs need to possess a high non-linearity (NL). However, with a large challenge space (usually ≥ 64 bits), measuring NL by classical techniques like Walsh transformation is computationally infeasible. In this paper, we propose the usage of a heuristic-based measure called non-homomorphicity test which estimates the NL of Boolean functions with high accuracy in spite of not needing access to the entire challenge-response set. We also combine our analysis with a technique used in linear cryptanalysis, called Piling-up lemma, to measure the NL of popular PUF compositions. As a demonstration to justify the soundness of the metric, we perform extensive experimentation by first estimating the NL of constituent Arbiter/Bistable Ring PUFs using the non-homomorphicity test, and then applying them to quantify the same for their XOR compositions namely XOR Arbiter PUFs and XOR Bistable Ring PUF. Our findings show that the metric explains the impact of various parameter choices of these PUF compositions on the NL obtained and thus promises to be used as an important objective criterion for future efforts to evaluate PUF designs. While the framework is not representative of the machine learning robustness of PUFs, it can be a useful complementary tool to analyze the cryptanalytic strengths of PUF primitives.**

***Index Terms*—Strong PUFs, Non-linearity, Cryptanalysis, Cryptanalytic Attacks, Non-homomorphicity Tests.**

I. Introduction

PHYSICALLY Unclonable Function (PUF) is a hardware-based security primitive that leverages the intrinsic device properties to realize a pseudorandom instance-specific function [1], [2]. This makes it an important cryptographic primitive which is used as a building block in various security applications such as device identification [1], [3]–[5], authentication protocols [6], [7], random number generation [8], [9], advanced cryptographic protocols [10]–[15] and many more. PUFs are broadly categorized into two groups, namely Strong PUF and Weak PUF, on the basis of the number of admissible challenge-response pairs (CRPs) [16], [17]. Strong PUFs are characterized by a large number of CRPs and a public interface to query the PUF, i.e. a user should be able to feed any challenge and obtain its corresponding response. Also, given the public interface, the challenge-response set should be large enough, so that an adversary cannot obtain all possible CRPs. Some popular Strong PUF constructions include the Arbiter PUF [3], [18], XOR Arbiter PUF [3], Feed-Forward Arbiter PUF [18], [19], analog PUFs [20]–[22], SHIC PUF [23], [24], Lightweight Secure PUF [25], Bistable Ring PUF [26], Subthreshold PUF [27], Lattice PUF [28], Voltage Transfer PUF [29], and Multiplexer PUF [30]. On the other hand, Weak PUFs allow a relatively less number of CRPs and thus have a restricted querying interface [16], [17]. Some example Weak PUF constructions include Ring Oscillator PUF [3], SRAM PUF [8], DRAM PUFs [31], [32], butterfly PUFs [33], buskeeper PUFs [34], transistor PUFs [35], or diode PUFs [36].

The above-mentioned features make Strong PUFs a prime candidate in several applications such as lightweight authentication [6], [37]–[41], advanced cryptographic protocols [10], [12], [42], [43], or hardware obfuscation [44], [45] that mandate cryptographic properties such as unpredictability, randomness and high non-linearity. Unpredictability of Strong PUFs has been assessed using attack strategies ranging from classical cryptanalytic techniques [46]–[48] to machine-learning (ML) based modeling attacks [49]–[54]. In ML-based attacks [49], [50], [53], [55], an adversary eavesdrops on a set of CRPs and feeds it to a suitable machine learning algorithm that produces a software model emulating the target PUF instance. The goal of the attacker is to predict the response for an unseen challenge with high accuracy. On the contrary, in case of a classical cryptanalytic attack, the objective is to exploit cryptographic weakness in the construction to launch a computational attack. A successful classical cryptanalytic attack on Strong PUF implies that an attacker can predict the response for an unknown challenge with probability greater than that of a random guess. Although there has been significant development in cryptanalysis of classical standard cryptographic primitives such as ciphers or Boolean functions such as Substitution Box (SBox), the assessment of PUFs from a cryptographic perspective lacks depth. Consequently, several popular PUF constructions do not have good cryptographic properties such as unbiasedness, nonlinearity or strict avalanche criteria (SAC), making them susceptible to attacks [46]–[48]. Although it might be infeasible to launch a pure cryptanalytic attack on PUFs due to its large challenge-response space, several attacks have amalgamated cryptanalysis with ML to launch successful attacks on standalone constructions [30], its compositions [46], [54] as well as protocols employing PUFs.

One of the common cryptanalytic attacks is the linear approximation attack, wherein an adversary tries to create a linear approximation of the primitive and uses the approximation to predict the output for an unknown input [56]. Let us consider the following example: Let l be the best linear

approximation of a function and if an adversary knows the outputs $l(\mathbf{a})$ and $l(\mathbf{b})$ for two random inputs $\mathbf{a}$ and $\mathbf{b}$, then it can determine the output of $\mathbf{a}+\mathbf{b}$ by computing $l(\mathbf{a})+l(\mathbf{b})$ (with some approximation error, depending on the distance between the target function and its best linear approximator), without querying the primitive. This allows the attacker to generate outputs for unknown inputs without querying the PUF, thereby compromising the security of authentication protocols.

On the same lines, a linear approximation-based attack was demonstrated on Multiplexer PUF (MPUF), wherein an attacker tries to approximate the final response with an internal APUF response bit. This is based on the crucial observation that the NL of the MUX Network is not 0.5. For $(n,3)$-MPUF (comprising of 8 data-line APUFs and 3 select-line APUFs), the bias in NL is calculated to be 25% which implies that the MPUF response differs from one of the data-line APUF outputs for only $\frac{1}{4}^{th}$ of the entire CRP set. Thus, an attacker can approximate a constituent APUF output using the final MPUF response with an error of only 25% and model it using standard ML algorithms [30]. The same procedure is repeated with a different set of CRPs to obtain models for each data-line APUF. Thus, it is crucial to choose a combiner function that has a high NL, although, this does not consider the NL of the constituent PUFs.

In this work, we focus on the cryptographic NL of PUFs and their XOR compositions. We present a systematic method to quantify the NL of individual instances efficiently using the SQnL framework. The overview of the nonlinearity assessment framework is depicted in Fig. 1. Since the functionality of every instance is uniquely determined by the device's intrinsic properties, each PUF instance realizes a different Boolean function. Thus, there can be some instances which have low NL and are therefore susceptible to classical cryptanalytic attacks. Thus, it is imperative to assess each instance individually to filter out/isolate/identify the *bad* instances. Classical techniques such as Walsh-Transformation are a misfit for Strong PUFs with standard challenge length of 64-bits or higher. This necessitates the formulation of a standard metric that can estimate the NL of Strong PUF constructions given black-box access to the PUF instance under test, by observing a significantly small fraction of CRPs. This technique has been extensively used in the field of property testing. Here we use one such test known as homomorphicity test to formulate a metric that can approximate NL with some error. The approximation error can be considered as a trade-off for less computational overhead. Increasing the number of CRPs provides a better estimate of NL at the cost of higher computational overhead. The NL computed by the proposed method does not correspond to the NL required to bolster security against ML-based modeling attacks. This test helps designers to evaluate specific instances for weakness concerning classical cryptanalytic attacks (such as linear cryptanalysis), and thus forms an essential part of the post-silicon PUF assessment tool set.

A. Contributions

The contributions of this work are as follows:

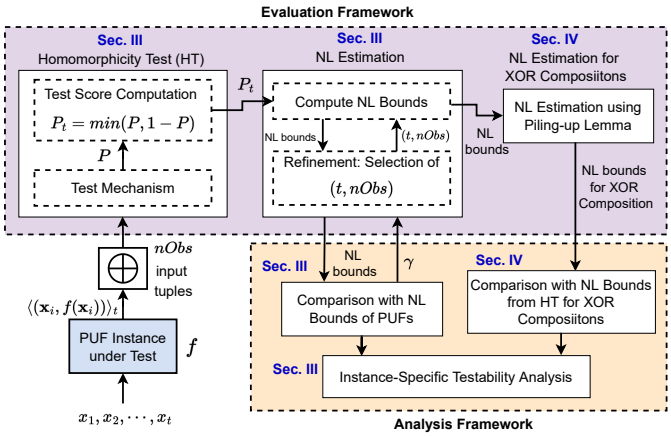


Fig. 1: Overview of SQnL Framework

- We develop a computationally efficient, easy-to-perform test to estimate the NL of Strong PUF constructions. This does not require any knowledge about the construction and provides a good estimate of the NL with a significantly small fraction of the entire challenge-response set. The test checks the failure probability of the homomorphicity property over a set of randomly chosen inputs, hence termed the PUF Homomorphicity test.
- We present a guideline for the choice of parameters for the PUF homomorphicity test and explain with the help of a representative example of XOR Arbiter PUF.
- We apply the test to popular Strong PUF constructions such as XOR Arbiter PUF, Feed-Forward Arbiter PUF and XOR Bistable Ring PUF for different test parameters. We evaluate the accuracy of the test by comparing it with the NL results obtained from the Sagemath tool. We demonstrate empirically that the actual nonlinearity lies within the NL bounds computed from the homomorphicity test for PUFs with small challenge lengths.
- We perform instance-specific analysis for the above-mentioned PUF families over 100 random PUF instances. Using empirical results, we depict how the estimated bounds capture the variation in NL across different instances. We discuss the implications of this test on the *bad* instances of PUF and how the test can identify such bad instances.
- We propose a mechanism to extend the NL estimation for XOR compositions of PUFs using a standard technique called Piling-up lemma [56]. We empirically demonstrate the efficacy of the heuristic in estimating NL of k-XOR APUF for $k = 2-8$.
- We consolidate the test parameters for which the PUF homomorphicity produces the most accurate estimation over 200 randomly simulated PUF instances of Arbiter PUF, XOR Arbiter PUF, Feed-Forward Arbiter PUF, Bistable Ring PUF and XOR Bistable Ring PUF.

The proposed test estimates the NL of Strong PUFs from a black-box perspective, thereby assessing the resilience against traditional cryptanalysis attacks performed on cryptographic primitives such as block ciphers. Note that the NL estimated

by the test does not correspond to the NL exploited by ML attacks.

B. Related Works

Ever since the emergence of PUF, it has been evaluated from the perspective of Boolean functions [57]–[60]. In [57], the authors propose a tool that assesses the robustness of PUFs against provable ML attacks. It uses metrics such as noise sensitivity and average sensitivity and property testing algorithms to assess the vulnerability of a PUF construction to a Fourier-based learning attack, also known as the LMN attack [61]. In this attack model, an attacker creates a hypothesis of the target PUF using solely the Fourier coefficients of the low-degree terms. The tool takes a set of challenge response pairs, the required accuracy and confidence of the algorithm and outputs the number of CRPs required to learn the PUF design, given its noise sensitivity is low. Although this provides provable guarantees against ML attacks, the tool does not evaluate the robustness of construction to mathematical cryptanalytic attacks. In [60], the authors present a cryptographic evaluation of Arbiter PUF-based constructions. It also proposes a variant of APUF that achieves good cryptographic properties with respect to differential analysis. In the direction of testability of PUFs, [59] proposes a testability framework for PUFs that assesses the quality of a PUF design using correlation analysis and statistical tests. Some first steps in the direction of automatically assessing Strong PUF security against ML-attacks have already been made in [62]. Their work systematically monitors the sensitivity of Strong PUF responses against small perturbations of the applied challenge.

C. Organization of this Paper

The remainder of the paper is organized as follows. Sec. II presents the preliminary concepts required in this work. In Sec. III, we propose the PUF homomorphicity test followed by its empirical analysis on some popular Strong PUF architectures including Arbiter PUF, Feed-Forward APUF, Bistable Ring PUF and their XOR compositions. Sec. IV presents a mechanism to estimate the NL of XOR compositions of Strong PUF designs. Sec V concludes the work.

II. Preliminaries

In this section, we present a brief background of the concepts required in this paper.

A. Non-linearity of Boolean Functions

A n-variable Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ is said to be affine if it can be represented as

$$f(x_1, x_2, \cdots, x_n) = u_0 + u_1x_1 + u_2x_2 + \cdots u_nx_n$$

where $u_i \in \{0,1\}\ \forall i \in [n]$ are the coefficients and $u_0 = 1$. If $u_0 = 0$, then f becomes a linear function in $\mathbb{F}_2^n$. The coefficients collectively can be represented by a coefficient vector $\mathbf{u} = (u_0, u_1, u_2, \cdots, u_n)$.

Let A_2^n denote the set of all affine functions in $\mathbb{F}_2^n$. The NL of a Boolean function f, denoted by N_f is the distance to its nearest affine function in $\mathbb{F}_2^n$ and is given by

$$N_f = \min_{g \in A_2^n} \frac{d(f,g)}{2^n}$$

where d denotes the Hamming distance. It can be computed using the Walsh transform (discrete Fourier transform) as

$$N_f = \frac{1}{2} - \frac{1}{2} \max_{\mathbf{u} \in \mathbb{F}_2^n} |W_{\widehat{f}}(\mathbf{u})| \tag{1}$$

where $\widehat{f}$ is the sign function corresponding to f, defined as $\widehat{f} = (-1)^f$ and $W_{\widehat{f}}(\mathbf{u})$ denotes the Walsh transform of $\widehat{f}$ and is given by

$$W_{\widehat{f}}(\mathbf{u}) = \frac{1}{2^n} \sum_{\mathbf{x} \in \mathbb{F}_2^n} \widehat{f}(\mathbf{x})(-1)^{\mathbf{u}.\mathbf{x}} \tag{2}$$

Here, $\mathbf{u} \in \mathbb{F}_2^n$ denotes the coefficient vector corresponding to an affine function. Here N_f represents the normalized NL and it can take values in $[0, 0.5)$. The computational overhead involved in N_f calculation using Walsh transform is exponential in the number of variables n.

B. Affinity Test of Boolean Functions

One of the commonly used techniques for testing the affinity of a Boolean function is to check whether $f(\mathbf{x}+\mathbf{y}+\mathbf{z}) + f(\mathbf{x}) + f(\mathbf{y}) + f(\mathbf{z}) = 0$ holds for a set of inputs $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_2^n$ chosen uniformly at random. If the test passes for most of the input triplets, f is said to be affine.

A generalization of this test, known as the $t+1$-th order non homomorphicity test [63] evaluates the probability $P_t(f)$ that a function f fails the following test

$$f(\mathbf{x}_1+\mathbf{x}_2+\cdots+\mathbf{x}_t)+f(\mathbf{x}_1)+f(\mathbf{x}_2)+\cdots+f(\mathbf{x}_t) = 0 \tag{3}$$

for all input tuples $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t) \in (\mathbb{F}_2^n)^t$. For odd t, f is affine if and only if $P_t(f) = 0$. On the other hand for even t, f is affine if and only if $P_t(f) \in \{0,1\}$ while f is linear if and only if $P_t(f) = 0$.

The fail probability $P_t(f)$ is an indicator of NL of a function f. The relationship between $P_t(f)$ and NL is established in [64]. It is shown that for odd $k \geq 3$, $P_t(f)$ is bounded as follows

$$\frac{1}{2}\Big(1-(1-2N_f)^{t-1}\Big) \leq P_t(f) \leq \frac{1}{2}\Big(1-(1-2N_f)^{t+1}\Big) \tag{4}$$

From Eq. 4, we obtain the following bounds on NL of a Boolean function in terms of $P_t(f)$

$$\frac{1}{2}\Big(1-\sqrt[t+1]{1-2P_t(f)}\Big) \leq N_f \leq \frac{1}{2}\Big(1-\sqrt[t-1]{1-2P_t(f)}\Big) \tag{5}$$

C. Relationship between NL and Fourier Spectrum

Non-linearity of a Boolean function is closely related to its Fourier spectrum, particularly the largest Fourier coefficient in the expansion. For Fourier analysis, a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ is generally represented as $f : \{-1,+1\}^n \rightarrow \{-1,+1\}$ using the encoding $\chi(x) = (-1)^x$. The Fourier expansion of $f : \{-1,+1\}^n \rightarrow \{-1,+1\}$ is a multilinear polynomial that represents f in terms of all the n-bit linear functions χ_S. This is mathematically represented as

$$f(\mathbf{x}) = \sum_{S \subseteq [n]} \widehat{f}(S).\chi_S(\mathbf{x}) \tag{6}$$

where $\chi_S(\mathbf{x}) = \prod_{i \in S} x_i$ represents a linear function [1], S denotes the set of input variables that contribute to the corresponding linear function. The Fourier coefficient given by $\widehat{f}(S) = \langle f, \chi_S \rangle = \mathbb{E}_{\mathbf{x} \in \{-1,+1\}^n}[f(\mathbf{x})\chi_S(\mathbf{x})]$ represents the distance of f to χ_S. The Fourier coefficient can be related to the Hamming distance to the corresponding linear function as $\widehat{f}(S) = 1 - 2\mathsf{HD}(f, \chi_S)$ where HD denotes the Hamming distance. Since the set of affine and linear functions are complementary, we can directly compute NL from the Fourier expansion.

For Bent functions, the class of Boolean functions with the highest NL, all the Fourier coefficients have the same magnitude, implying it has equal distance from all the linear functions. On the other hand, for a generic Boolean function, the fail probability of the affinity/homomorphicity test P_t is bounded by the largest coefficient in its Fourier spectrum [65].

D. Piling-up Lemma

Piling-up lemma [56] is a technique used in linear cryptanalysis to compute linear approximation of Boolean functions. It states that the output bias of a linear Boolean function is related to its input bias, given that the inputs are independent Boolean variables. Mathematically,

$$\epsilon(x_1 \oplus x_2 \oplus \cdots \oplus x_k) = 2^{k-1} \prod_{i=1}^{k} \epsilon_i \tag{7}$$

where $x_1, \cdots, x_k \in \{0,1\}$ denote independent binary random variables and $\epsilon_i = p_i - \frac{1}{2}$ denotes the bias or deviation of the probability p_i of x_i from $1/2$.

E. Physically Unclonable Functions

A Physically Unclonable Function (PUF) is a pseudo-random function realized from the intrinsic physical characteristics of the underlying device. For instance, PUFs instantiated on silicon chips leverage the inherent physical features of the IC that occur due to uncontrollable variations in the manufacturing process and are known as Silicon PUFs. We briefly describe some of the Silicon PUF constructions, namely Arbiter PUF, Feed-Forward Arbiter PUF, Bistable Ring PUF and their XOR compositions.

[1]The AND operation in $\{-1,+1\}$ domain corresponds to an XOR operation in $\mathbb{F}_2$.

1) Arbiter PUF and XOR Arbiter PUFs: Arbiter PUF (APUF) [18] is the seminal primitive PUF architecture that utilizes the delay difference of two symmetrically designed parallel delay lines to generate a random and unique response. In an ideal scenario, the delay difference should be zero between two symmetrically laid-out paths. However, in practice, it is non-zero due to uncontrollable variation in the IC manufacturing process that introduces random offset between the two delays.

An APUF comprises n two-port path-swapping switches connected in serial. The input challenge bits are used as the control to the switches that determine which path (straight or cross) will be selected for the input signal's propagation. The two input terminals of the first stage are connected to a common end, to which an excitation signal is applied. The arbiter (D flip-flop) at the end of the two paths returns a 1-bit response (logic-0 or logic-1) depending on which of the two paths is faster. Fig. 2a depicts a n-stage APUF that takes a n-bit challenge $\mathbf{c} = (c_1, c_2, \cdots, c_n)$, where c_i is the control signal to the i^{th} switch.

However, APUFs have been shown to be vulnerable against modeling as well as reliability attacks [49], [53]. Therefore, to strengthen the APUF construction against such attacks, XOR Arbiter PUFs (XOR APUF) [3] were proposed. A k-XOR APUF has k individual APUFs whose outputs are combined using an XOR gate to produce a global 1-bit response as shown in Fig. 2d.

2) Feed-Forward Arbiter PUF and Feed-Forward XOR Arbiter PUFs: Feed-Forward Arbiter PUFs (FF-APUF) [19] uses an internally generated signal to drive one or more switches in the APUF delay chain instead of the external challenge bits. The rationale behind the construction is to obfuscate a part of the actual challenge fed to the PUF to enhance its resilience to ML attacks. The internally generated signal is a function of the delay differences accumulated in earlier parts of the circuit and is obtained from an additional arbiter as depicted in Fig. 2b. The delay paths emanating from a intermediate stage in the delay chain is fed to an additional arbiter, whose outcome is provided as a challenge bit to a succeeding stage, thereby creating a feed-forward loop. The stage whose emanating signals are fed to the arbiter determines the start of the loop and the stage whose challenge bit is replaced by the arbiter output specifies the end of the loop. A FF-APUF can comprise of more than one loops. The number of loops, the start and end position of the loops and the amount of overlap between loops can give rise to different types of feed-forward Arbiter PUF architectures. FF-APUFs with multiple loops can be categorized on the basis of the position of loops into two classes, namely FF-APUF with non-overlapping loops and FF-APUF with overlapping loops.

The addition of the feed-forward loop makes the arbiter PUF non-differentiable, thereby making it robust against linear ML attacks using algorithms such as Linear Regression (LR) and Support Vector Machine (SVM) barring a few cases where the number of non-overlapping loops are small. The resilience of FF-APUF to ML attacks can be improved further by using XOR FF-APUFs. A k-XOR FF-APUF has k independent FF-APUFs whose outputs XORed to obtain the final PUF

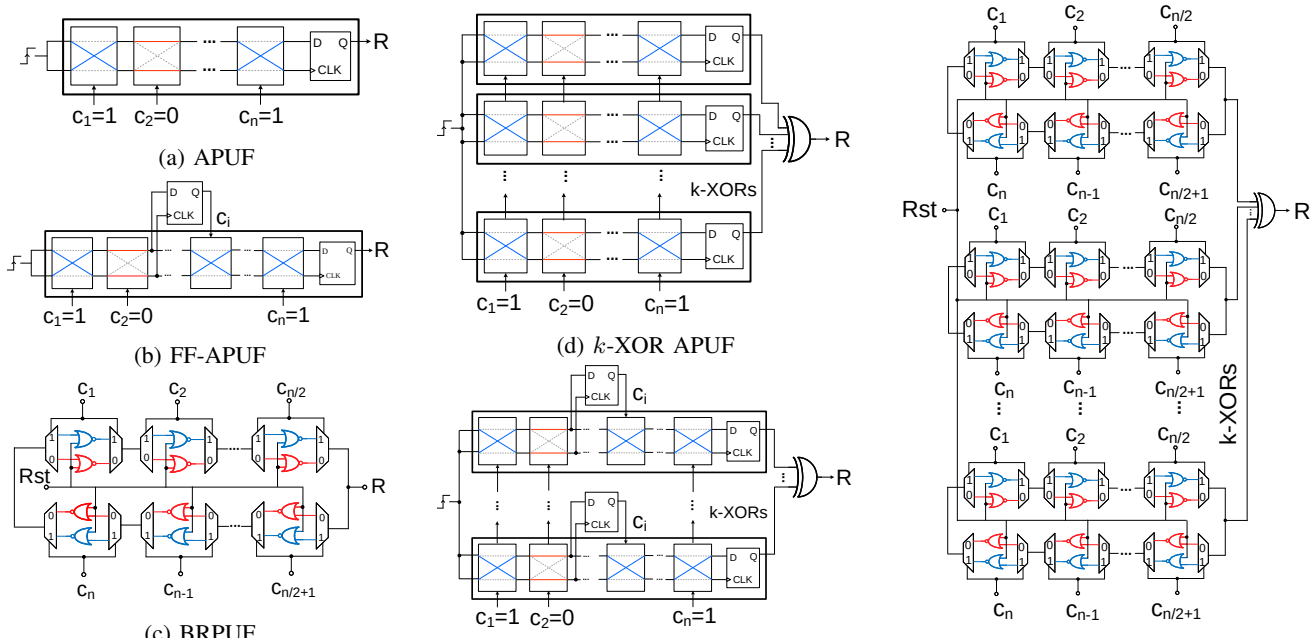


Fig. 2: Schematic Diagram of Strong PUF constructions

response.

3) Bistable Ring PUF and XOR Bistable Ring PUFs: A Bistable Ring PUF (BRPUF) comprises of an even number of inverting cells connected serially in a loop as depicted in Fig. 2c. It generates a random response from either of the two stable states (010101 . . . or 101010 . . .) of the loop [26], [66]. BRPUF considers the challenge-dependent delay of each inverting stage to generate a stable response.

Each stage of BRPUF has two inverting delay elements with MUXs to select the challenge-driven signal path. The linear model for BRPUF, first proposed in [66] uses an additive delay model similar to APUFs as given below.

$$r = sgn\left(\sum_{i=1}^{n} \alpha_i + c_i\beta_i\right) \tag{8}$$

where c_i denotes the i^{th} challenge bit and the weight values (α_i, β_i) represent the difference between the pull-up strength and the pull-down strength in BRPUF as opposed to the stage delays in APUF. In order to increase the robustness of the construction against modeling attacks XOR BRPUF is proposed, wherein the final response is obtained XORing the outputs of k BRPUFs. Furthermore, the same linear-model is used in the PyPUF tool for the abstraction of BRPUFs and XOR-BRPUFs.

F. Simulation of Challenge-Response Pairs

PyPUF [67] is an open-source library which provides tools to simulate, analyse as well as model different PUF designs. This library uses Linear Threshold Functions (LTFs) to model different delay-based PUFs [51]. Arbiter PUF and its compositions namely XOR-APUF, XOR Feed-Forward APUF are modelled using LTFs and model-specific parameters such as the number of XOR branches, starting and ending location of feed-forward loop etc.

```
>>> from pypuf.simulation import XORArbiterPUF
>>> puf = XORArbiterPUF(n=64, k=8, seed=1, noisiness
    =.05)
>>> from pypuf.io import random_inputs
>>> puf.eval(random_inputs(n=64, N=3, seed=2))
array([-1, -1,  1], dtype=int8)
```

Listing 1: XOR Arbiter PUF Instantiation

Each bit of the challenge in the PyPUF tool is generated from a uniform distribution. A challenge transformation is performed on the raw challenge input to get a transformed challenge with parity bits based on the PUF model (e.g. Arbiter PUF). The LTF generates weights that are sampled from a Gaussian distribution, which are then used to evaluate the dot-product with the transformed challenges. PyPUF tool uses the sign of the LTF output for Arbiter PUF. In the case of XOR Arbiter PUFs, the output of t LTFs are multiplied and the sign of the output is the final PUF response. There is also a noise parameter in the PyPUF tool which determines the standard deviation of the noisy random variable added to the LTF used for noisy simulations. Listing 1 and 2 shows sample instantiations for Arbiter PUF and XOR Arbiter PUF respectively.

III. Non-Linearity Evaluation of PUFs

Non-linearity is an essential property required in cryptographic primitives. In this section, we propose a test to evaluate the NL of a PUF instance. We first present the test methodology and elaborate the test parameters and results using examples of Arbiter PUF, Feed-Forward Arbiter PUF, Bistable Ring PUF and its XOR compositions.

A. Test Methodology and Test Rationale

Non-linearity computation of a PUF instance using classical techniques such as Walsh Transformation requires the entire challenge-response set, making it computationally infeasible for an instance with a standard challenge length of 64 bits and higher. It necessitates an efficient approach for NL computation of PUF constructions using a small fraction of the entire challenge-response set.

Test 1 (PUF HOMOMORPHICITY TEST)**.**

Under Test:

- PUF-instance with binary challenges $\mathbf{c}_i = b_i^1 \cdots b_i^n$ of length n, and with single-bit responses R_i.

Test Score/Output:

- A number $P_t \in [0, 1]$ that estimates the probability with which the instance fails the homomorphicity test. The probability is calculated over $nObs$ number of t-input tuples, where a t-input tuple comprises of t randomly generated challenges $(\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_t)$.

Test Variants and Practical Application:

- In practice, for a given t, P_t can be estimated by the following method:
  - A set of t n-bit challenges $\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_t$ is chosen uniformly at random. All the challenges are fed to the PUF instance and their responses are XORed to obtain a single bit output, say R.
  - A bitwise XOR of the t challenges is performed to obtain another n-bit challenge $\mathbf{c}'$. This challenge is applied to the PUF-instance and let the corresponding response be represented by R'.
  - The former two steps are repeated for $nObs$ input tuples. P_t is calculated as the fraction of tuples for which R and R' differ.
- For a given t, P_t value is unique to a PUF-instance. Ideally, all possible t-tuples, i.e. $nObs = \binom{2^n}{t}$ should be considered. However, $nObs \geq 10^6$ gives a decent estimate of NL. We explain the role of $nObs$ in NL estimation in the following subsection.
- The choice of t is crucial to get a correct estimation of NL. Based on our experiments, we observe that one should start with $t = 3, 5$. A higher t value gives a better estimate. However, the permissible t values depend on the number of tuples $nObs$.

Ideal Test Score and Simple Test Interpretation:

- The ideal value of the test score is 0.5, which implies that the instance under test has an NL of 0.5. Any deviation from the ideal value implies that the NL is less than 0.5. Although the amount of deviation of NL from 0.5 depends on the choice of t, as given in Eq. 5. Even a small deviation of P_t from 0.5, can imply a significantly low NL.

To estimate the NL of the PUF-instance under test, we compute $min(P_t, 1 - P_t)$ and substitute the obtained value in Eq. 5. It returns an upper and lower bound of NL, depending on the value of t. Note that, $min(P_t(f), 1 - P_t(f))$ ensures that the value substituted in Eq. 5 can take a maximum value of 0.5, as a value greater than 0.5 of $P_t(f)$ implies a much lower $P_t(f')$ [2]. In NL estimation, we take the minimum of $P_t(f)$ and $P_t(f')$ as it takes into account the distance of the best linear approximator from f and f'. For an affine function, since $P_t(f) = 0$ irrespective of the value of t (odd t) (refer Sec. II-B), the lower and upper bound converge at $N_f = 0$. However, for a non-affine function, the choice of t and $nObs$ play a crucial role in the homomorphicity test and consequently in NL estimation. Fig. 3 illustrates the relationship between P_t and the NL upper bound obtained from Eq. 5 for different t, from which we make the following observations:

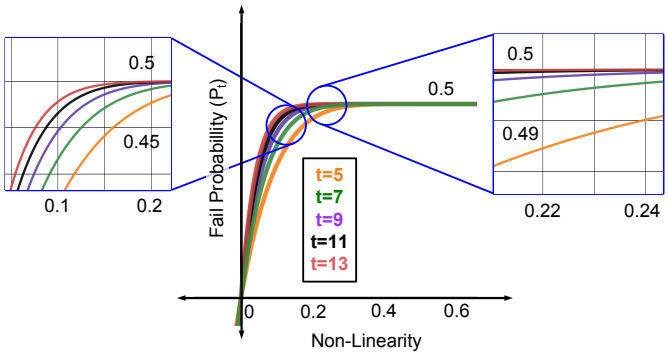


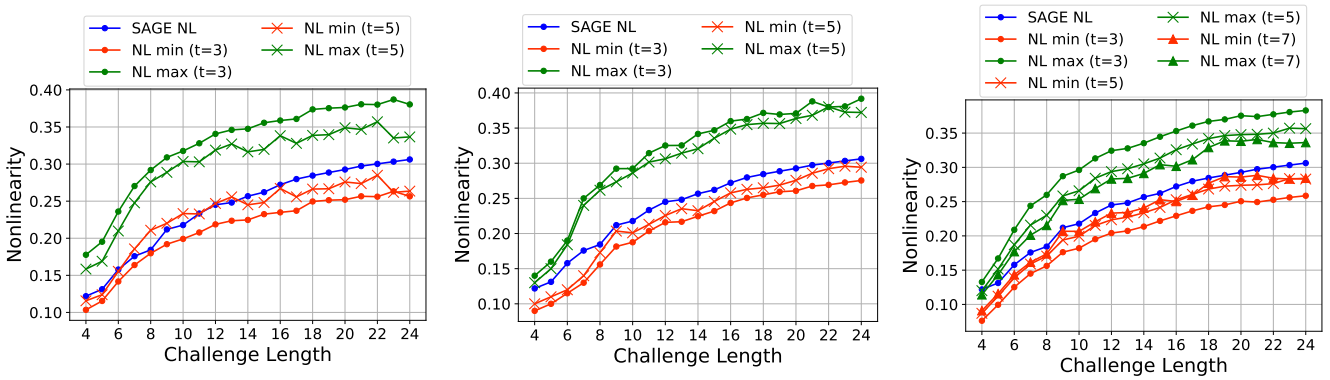
Fig. 3: Flip Probability vs Non Linearity

1) As t increases, the slope of the curve gets steeper. So, for a PUF-instance under test (having a fixed NL), the P_t gets closer to 0.5 on increasing t. Let us refer to the vertical grid line in the zoomed part on the right in Fig. 3. For 0.24 NL, the point of intersection of each curve to the vertical line gets closer to 0.5 as t increases. Moreover, if the NL of the PUF-instance is high (close to 0.5), the separation between the curve and 0.5 diminishes rapidly on increasing t. In other words, the fail probability deviates from 0.5 by a very small fraction.
2) For a given fail probability value corresponding to a PUF-instance under test, the estimated NL reduces as t increases. This can be understood better by following a horizontal line in the left zoomed area in Fig. 3. With an increase in t, the point of intersection shifts closer to zero, resulting in an underestimation of NL.

Given these insights, we elaborate on the significance of t and $nObs$ in the homomorphicity test as follows.

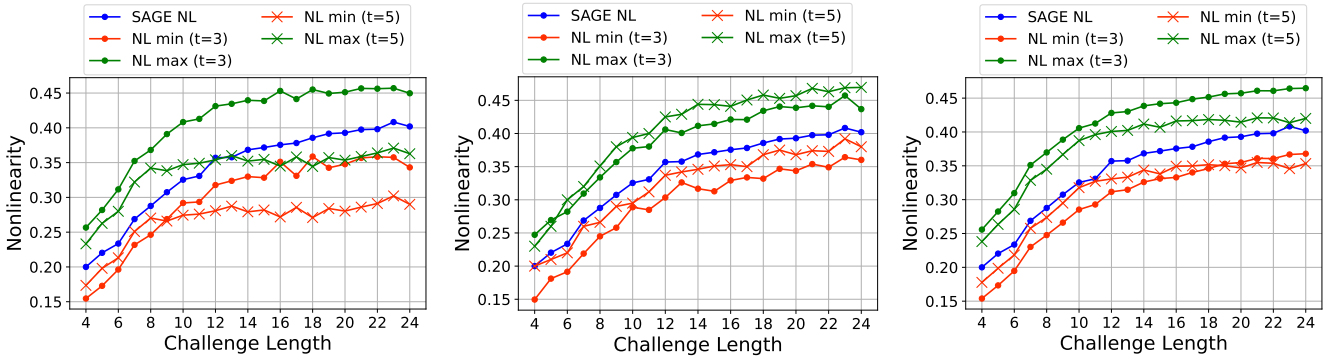
Impact of t on non-linearity estimation: From Eq. 5, we understand that increasing t results in a better NL estimation as the interval between the upper and lower bounds reduce. Intuitively, the homomorphicity test becomes more *strict/stringent* as t increases, thereby increasing the fail probability P_t. Thus,

[2] f' denotes the complement of f. If f has a fail probability $P_t(f)$, then the fail probability of f' is $P_t(f') = 1 - P_t(f)$.



(a) $nObs = 10K$ (b) $nObs = 100K$ (c) $nObs = 1M$

Fig. 4: Non-linearity of APUF estimated using different $nObs$ and t



(a) $nObs = 10K$ (b) $nObs = 100K$ (c) $nObs = 1M$

Fig. 5: Non-linearity of 2-XOR APUF estimated using various $nObs$ and t

for an accurate NL estimation, t should be high which in turn requires the fail probability to be very precise (from Observation 1)). Note that, one can choose a smaller value of t, for which the fail probability need not be calculated very accurately, however, the resultant NL bound interval would be wide.

Impact of $nObs$ on the non-linearity estimation: For a given t, the $P_t(f)$ is basically the fraction of input tuples for which the homomorphicity test fails. Thus the precision of P_t is determined by $nObs$. Although it is computationally infeasible to consider all possible t-input tuples for the test, we can work with a significantly lower number of tuples. To select a higher value of t, one needs to calculate the fail probability accurately, therefore $nObs$ should be high. A smaller $nObs$ value will result in underestimation of the fail probability (rounded up to less number of decimal places), which will in turn generate a much lower NL estimate (refer to Observation 2)).

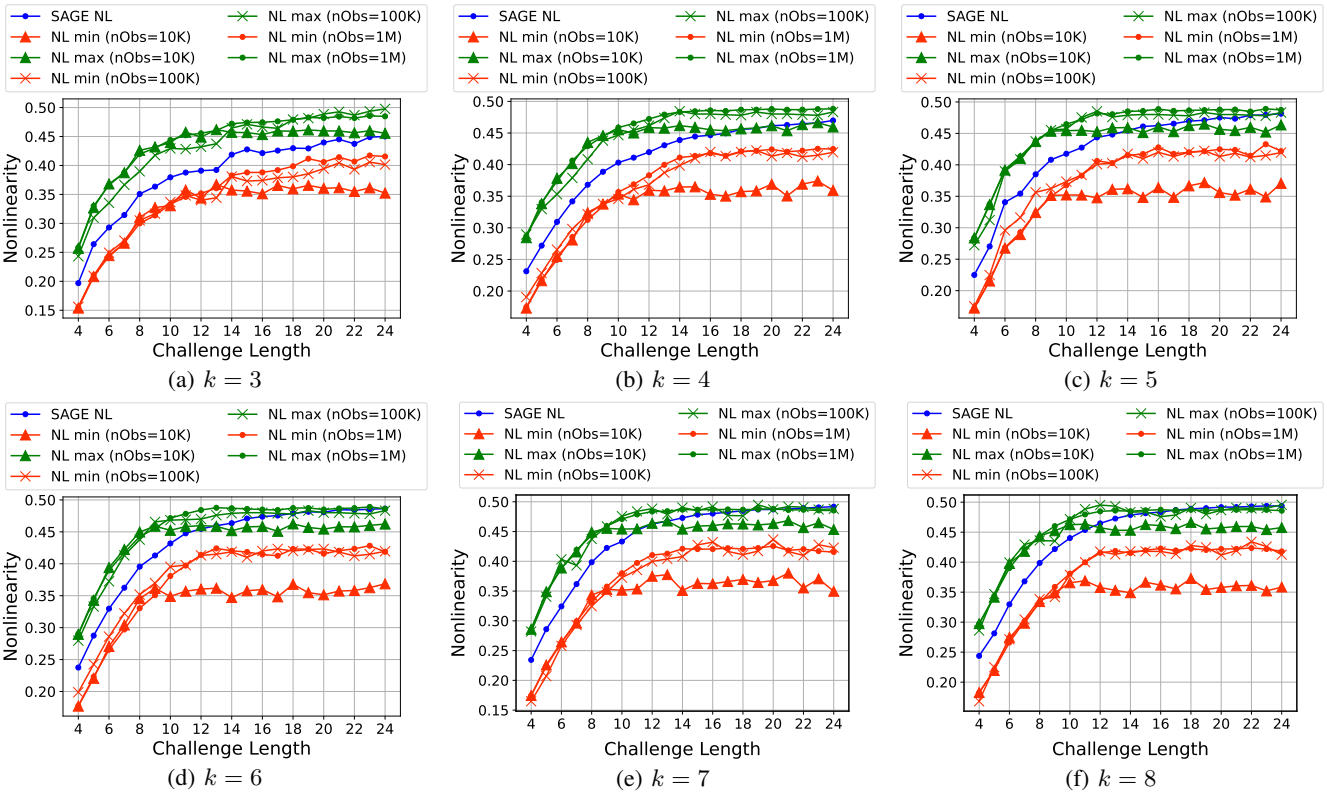
In the next section, we illustrate this phenomenon with the help of XOR APUF, FF-APUF and BR-PUF constructions. First, we demonstrate the impact of $nObs$ on the permissible values of t and the accuracy of the NL estimation. For this, we compare the NL bounds obtained for XOR APUF for different $nObs$ and t with actual NL obtained from Sagemath tool [68]. We perform this analysis on PUFs with small challenge lengths. Next, using FF-APUF and BR-PUF designs, we empirically determine suitable t for which the estimated NL approximates the actual NL accurately. Based on our analysis we extend the PUF homomorphicity test for large challenge lengths in the following section.

B. Non-Linearity of XOR Arbiter PUFs

We perform the PUF homomorphicity test on 20 randomly chosen instances of APUF and k-XOR PUF ($k = 2, 3, 4, 5, 6$) that take challenges of length $n = 4, 5, \cdots, 24$ for t in $\{3, 5, 7, 9\}$ and $nObs = 10K, 100K$ and $1M$. The NL bounds obtained from the test are verified against the NL obtained from the Sagemath tool [68]. Fig. 4 depicts the NL bounds averaged over 20 APUF instances using $10K, 100K$ and $1M$ tuples, from which we make the following observations:

- For $nObs = 10K$, the homomorphicity test correctly estimates the NL only for $t = 3$. For a higher value, say $t = 5$, the test underestimates the NL bounds. This can be attributed to low precision of P_t.
- As $nObs$ increases, the homomorphicity test performs well for $t = 5, 7$.
- The accuracy of the homomorphicity test enhances as t increases.

We perform the same analysis for 2-XOR APUF and demonstrate the results in Fig. 5. Similar to APUF, given $10K$ input tuples, the homomorphicity test performs well



(a) $k = 3$ (b) $k = 4$ (c) $k = 5$

(d) $k = 6$ (e) $k = 7$ (f) $k = 8$

Fig. 6: Non-linearity of k-XOR APUFs ($k = 3, 4, 5, 6, 7, 8$) estimated using different $nObs$ for $t = 3$

for $t = 3$. On the other hand, the test fails to estimate the NL correctly when 5-input tuples are considered. For a correct estimation, the user needs to increase $nObs$. For $nObs = 1M$, we observe that the actual NL lies within the estimated bounds. To experimentally demonstrate the impact of $nObs$, we perform the homomorphicity test on 20 instances of each k-XOR APUF ($k = 3, 4, 5, 6, 7, 8$) with $t = 3$ and vary $nObs = 10K, 100K$ and $1M$. We validate the bounds with NL values obtained from Sagemath as shown in Fig. 6. One key point to be noted is that the estimated bounds follow the trend of actual NL for higher $nObs$ as the challenge length increases. This crucial observation enables us to extend this test for PUFs with large challenge lengths. Going forward, we use $nObs \geq 1M$ for the homomorphicity test on other PUF designs. The choice of $nObs$ is also dependent on the challenge length.

To illustrate the change in NL of k-XOR APUF with change in k, we plot the NL of 100 instances of 20-bit APUF and k-XOR PUF where $k = 2, 3, 4, 5, 6, 7, 8$ (refer Fig. 7). The 100 instances are chosen uniformly at random and their NL is computed using the Sagemath tool. Fig. 7 clearly depicts the increase in average NL with an increase in k. Herein, we also observe that for APUF and 2-XOR APUF, the NL of individual instances vary significantly, while the variation reduces for higher values of k, as clearly depicted in Fig. 8. This implies that we can easily identify less non-linear instances that are susceptible to cryptanalytic attacks. Additionally, the NL of the different k-XOR APUF instances is distinguishable up to $k = 4$. Among higher k-XOR APUF values, this distinction

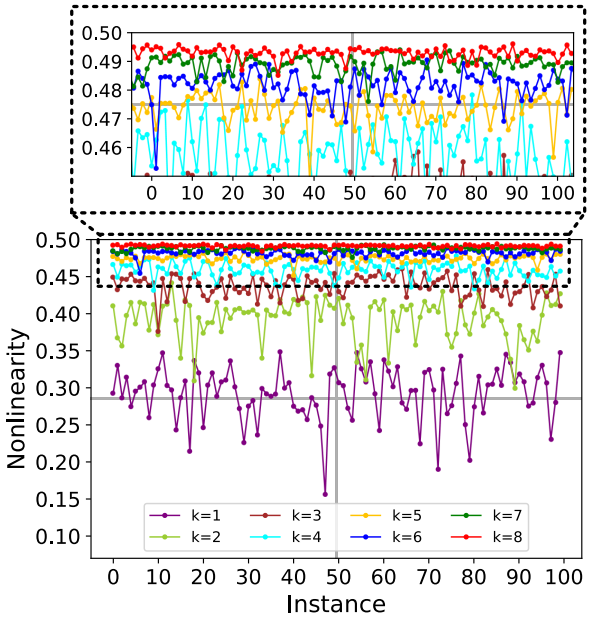


Fig. 7: Non-linearity of individual instances of 20-bit k-XOR APUF instances obtained from SAGE ($k = 1, 2, \cdots, 8$)

diminishes, implying that one can distinguish between a lower XOR APUF instance from a higher XOR APUF instance.

To evaluate the performance of the proposed test, we also plot the NL bounds estimated from the test along with the SAGE NL values for all instances in Fig. 9. Herein, we can observe that the estimated bounds follows the individual

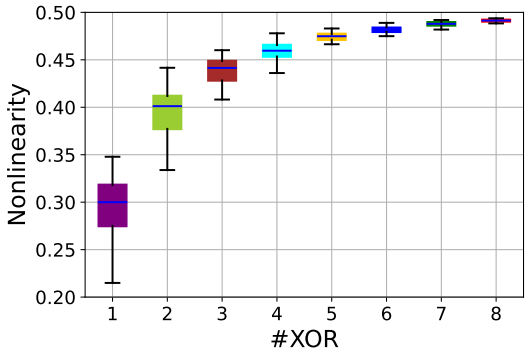


Fig. 8: Non-linearity Variance of individual instances of 20-bit k-XOR APUF instances obtained from SAGE ($k = 1, 2, \cdots, 8$)

NL closely, including the peaks and troughs (refer Fig. 9b). Moreover, the width of the bounds reduces as t increases. For higher k, there exist some instances for which the test estimates the NL incorrectly. We specify the fraction of instances for which the homomorphicity test computes NL range correctly using a parameter γ in each of the plots. We observe that as k increases, the value of γ reduces, which indicates an underestimation of NL due to less number of input tuples. To validate our conjecture, we re-evaluate γ for a larger $nObs$ value. We depict the results of the test for $nObs = 10M$ for k-XOR APUF ($k = 6, 7, 8$) in Fig. 9.

C. Non-Linearity of Feed-Forward Arbiter PUFs

We apply the same test to Feed-Forward Arbiter PUF (refer Sec. II-E). The rationale behind the FF-APUF design is to increase the NL of the construction, thereby increasing robustness against model-building attacks. To the best of our knowledge, no previous work in the literature has quantified the increase in NL due to the addition of feed-forward loops. For this work, we consider FF-APUF with non-overlapping as well as overlapping FF-APUF. In both configurations, we vary the loop length (denoted by l) and number of loops, depending on the challenge length. Additionally, in an FF-APUF with overlapping loops, we also vary the overlap in the loops, denoted by o. For a given FF-APUF construction, all loops have the same length and the loop structure spans the entire delay chain. For instance, let us consider a 20-bit FF-APUF with non-overlapping loops of length 4. The loop structure in this construction is $(1, 5), (6, 10), (11, 15), (16, 20)$. This representation specifies the starting and ending of each loop in terms of the challenge bit, i.e the first entry in each parenthesis denotes the challenge bit after which the the top and bottom signals are branched out to an arbiter and the last entry denotes the challenge bit where the arbiter output is fed. Similarly, we represent the loop structure of a 20-bit FF-APUF with overlapping loops of length 5 and an overlap of 2 as $(1, 6), (4, 9), (7, 12), (10, 15), (13, 18)$. Although the delay chain comprises of 20 stages the loop structure spans upto the 18^{th} stage as no more loops of length 5 can fit beyond that.

FF-AFPUF with nonoverlapping loops: First, we perform the homomorphicity test on n-bit FF-APUF with non-overlapping loops where $n = 10, 12, \cdots, 24$ and loop length varies in $l = 3, 4, 5$. We start with a challenge length of 10 so that we obtain at least two loops for $l = 3$. We use $1M$ input tuples and vary $t = 3, 5, 7, 9$. Fig. 10 depicts the average NL obtained from Sagemath along with the NL bounds obtained from the test, from which we make the following observations:

- The NL of an FF-APUF is lesser compared to an APUF of same challenge length (ref. Fig. 4). It decreases further with an increase in the number of loops. This can be explained
- The NL values depict a zig-zag behaviour that can be attributed to the end position of the last loop. Let us consider Fig. 10a as an example. In this $n = 12, 14, 16$ break the monotonic increase of NL as observed in case of APUF. In particular, the NL values are in the order $NL_{16} < NL_{12} < NL_{14}$. The loop structure for $n = 12, 14$ is $(1, 4), (5, 8), (9, 12)$ and for $n = 16$, the structure is $(1, 4), (5, 8), (9, 12), (13, 16)$. The lower NL for $n = 16$ as compared to $n = 12$ is due to more number of loops (4 and 3 respectively). Since $n = 12, 14$ have the same loop structure, the impact of increased challenge length is predominant. The remaining plots can be explained in a similar manner.

FF-AFPUF with overlapping loops: Next, we perform the test on FF-APUF with overlapping loops for $n = 10, 12, \cdots, 24$. The length of a single loop and the overlap between two consecutive loops represented as (l, o) can take values in $(3, 1), (4, 1), (5, 1), (5, 2), (6, 1), (6, 2)$. We use the same test parameters set for FF-APUF with non-overlapping loops. The test results are presented in Fig. 11, from which we make the following observations:

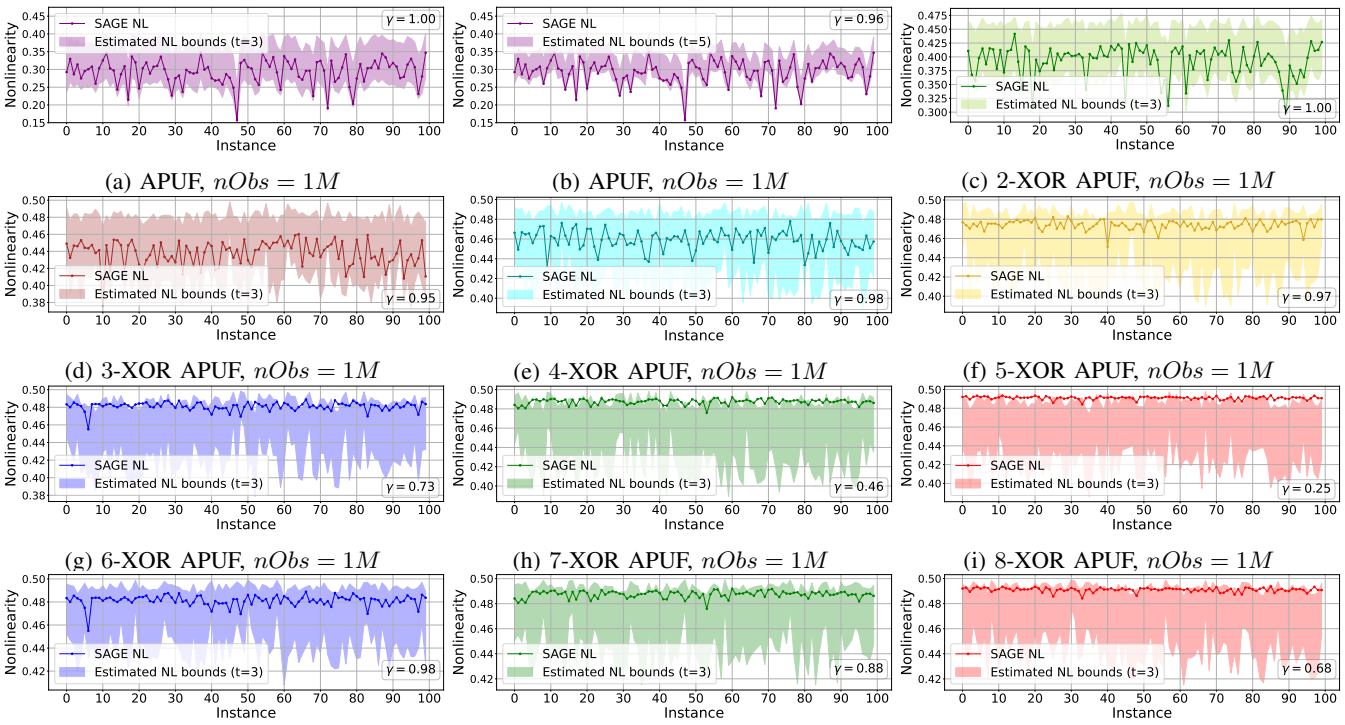
- The NL of FF-APUF with overlapping loops increases monotonically with the challenge length.
- The NL bounds determined from the homomorphicity test follows the SAGE NL pattern for $t = 3, 5$. For higher t values, the homomorphicity test underestimates the NL bounds, similar to an XOR APUF.

D. Non-Linearity of Bistable Ring PUFs

We apply the homomorphicity test to Bistable PUF and XOR Bistable Ring PUF (XOR BRPUF), which has a different internal structure than Arbiter PUF constructions. We perform the test over 20 randomly generated instances of each construction where k varies in $\{1, \cdots, 6\}$ using $1M$ observations, $pt = \{3, 5, 7, 9\}$ and present our results in Fig. 12. We observe the following:

- The rate of increase of BRPUF NL with respect to the challenge length is lesser than that of an APUF.
- For 1M observations, the homomorphicity test performs well for BRPUF and 2-XOR BRPUF. However, for higher values of k, the test underestimates the NL for k-XOR BRPUF.

To illustrate the change in NL with increase in number of XOR chains, we also plot the individual NL of each of the 100 20-bit XOR BRPUF where $k = 1, \cdots, 8$ (refer Fig. 14). From this plot, we observe significant variation in the NL of individual instances for BR-PUF and 2-XOR BRPUF.



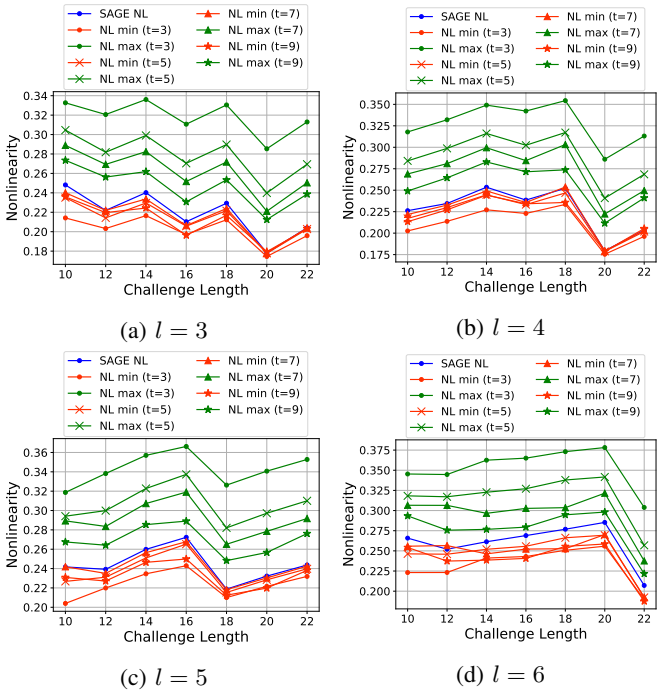
(a) APUF, $nObs = 1M$ (b) APUF, $nObs = 1M$ (c) 2-XOR APUF, $nObs = 1M$

(d) 3-XOR APUF, $nObs = 1M$ (e) 4-XOR APUF, $nObs = 1M$ (f) 5-XOR APUF, $nObs = 1M$

(g) 6-XOR APUF, $nObs = 1M$ (h) 7-XOR APUF, $nObs = 1M$ (i) 8-XOR APUF, $nObs = 1M$

(j) 6-XOR APUF, $nObs = 10M$ (k) 7-XOR APUF, $nObs = 10M$ (l) 8-XOR APUF, $nObs = 10M$

Fig. 9: Non-linearity estimation of individual instances of 20-bit XOR APUF



(a) $l = 3$ (b) $l = 4$

(c) $l = 5$ (d) $l = 6$

Fig. 10: Non-linearity of FF-APUF with non-overlapping loops estimated using $nObs = 1M$

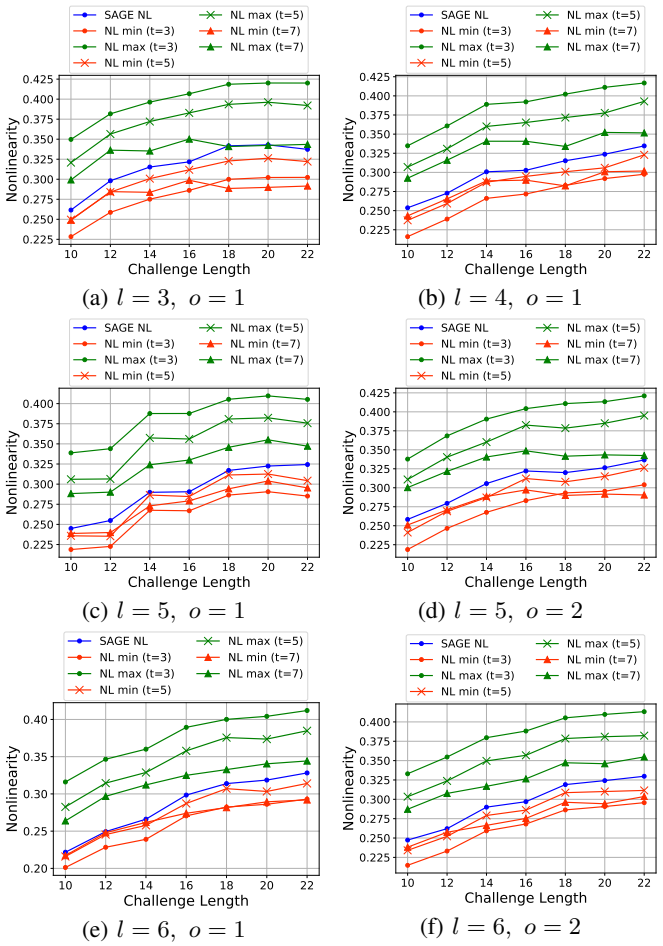
As k increases, the variation in NL of individual instances diminishes, as depicted in Fig. 13. Next, we plot the NL bounds estimated from the homomorphicity test along with the SAGE NL for all the 100 instances in Fig. 15. Herein, we denote the number of instances for which the proposed test estimates the NL correctly using γ. We observe the same phenomenon of decreasing γ in case of XOR BRPUF for higher values of k. Compared to XOR APUFs, we observe that the proposed test performs better for XOR BRPUF for higher values of k ($k = 6, 7, 8$).

E. Non-linearity Computation for standard challenge length PUFs

To assess the performance of the homomorphicity test on PUFs with standard challenge lengths, we perform the test on 20 instances of n-bit APUF and n-bit k-XOR APUF, where $n = 32, 64, 96, 128, 192$ and $k = 2, \cdots, 8$. We observe from the empirical results demonstrated in Fig. 16 that the NL of APUF and XOR APUFs increase almost monotonically with the challenge length. We perform the test by varying the number of input tuples from $100K$, $1M$ and $2M$. However, for $nObs = 1M$, we observe that the bounds were horizontal, thereby independent of the challenge length. On increasing $nObs$ to $2M$ and restricting tuple length to $t = 3$, we observe a monotonic increase in the NL with increasing challenge length. Additionally on increasing k, the estimated bound shifts close to 0.5, thereby illustrating NL enhancement due to XOR. Fig. 16 depicts the NL bounds obtained from the homomorphicity tests.

IV. Non-linearity Computation for PUF compositions using Piling-up Lemma

The results in the previous section establish the capability of the homomorphicity test in estimating the NL of several

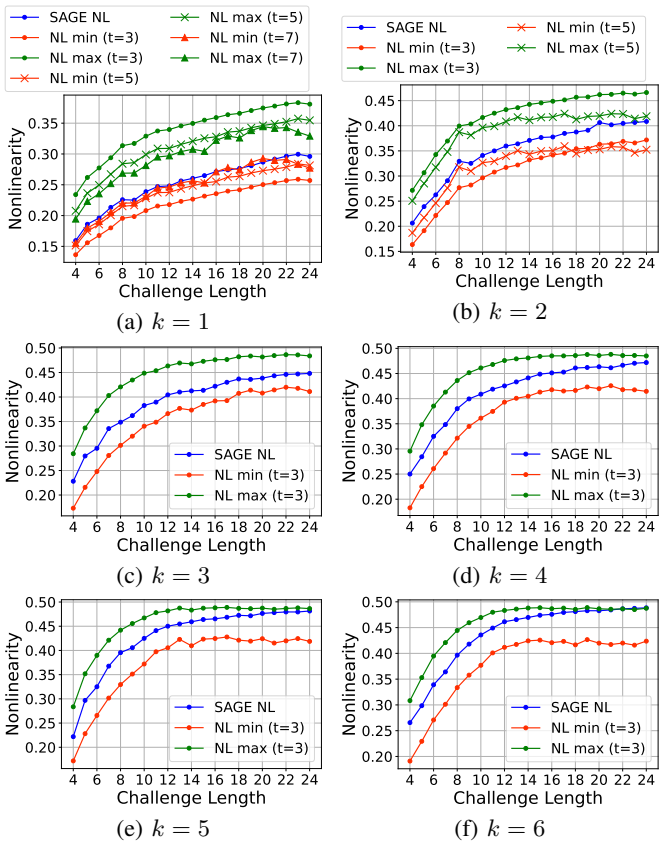


(a) $l = 3,\ o = 1$ (b) $l = 4,\ o = 1$

(c) $l = 5,\ o = 1$ (d) $l = 5,\ o = 2$

(e) $l = 6,\ o = 1$ (f) $l = 6,\ o = 2$

Fig. 11: Non-linearity of FF-APUF with overlapping loops estimated using $nObs = 1M$



(a) $k = 1$ (b) $k = 2$

(c) $k = 3$ (d) $k = 4$

(e) $k = 5$ (f) $k = 6$

Fig. 12: Non-linearity of k-XOR BRPUF estimated using $nObs = 1M$

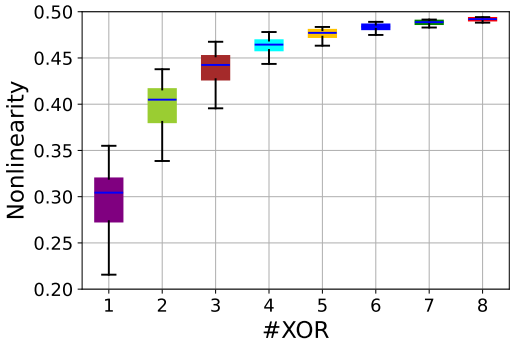


Fig. 13: Non-linearity of individual instances of 20-bit k-XOR BRPUF instances obtained from SAGE ($k = 1, 2, \cdots, 8$)

primitive PUF constructions. Although the test performs well on small XOR compositions, results presented in Sec. III-B and III-D depict that the proposed test underestimates the NL for higher values of k ($k = 7, 8$) (refer Tab. I). One way to address this is to increase $nObs$, however, the computation time increases linearly. In this section, we present a heuristic to extend the NL estimation for higher XOR compositions. To this end, we employ the Piling-up lemma (refer Sec. II-D) to explain the change in NL of k-XOR APUF, k-XOR BRPUF, with varying k. Although Piing-up lemma is traditionally used to estimate the bias of a linear Boolean function (XOR of independent Boolean variables) from the individual bias of the input variables, herein we employ this technique to compute an upper bound of NL of XOR of independent Boolean functions. We can assume the contributing Boolean functions (realized by constituent primitive PUFs) to be independent as the output of each of the primitive PUFs is determined from different delay parameters. The formulation of NL of XOR composition of PUF is explained as follows.

Let the NL of a PUF primitive taking an n-bit challenge be X_n. The bias of NL is the difference between the actual NL from its ideal value of 0.5 and is given by $\epsilon_n = 0.5 - X_n$. Assuming X_n to be a random variable, the NL of a k-XOR composition can be estimated as

$$X_n^k = \frac{1}{2} - 2^{k-1} \prod_{i=1}^{k} \epsilon_n \tag{9}$$

We assess the proposed heuristic using the following cases:
Case 1: All constituent PUF are affine. In this case $X_n = 0$, thus $X_n^k = \frac{1}{2} - 2^{k-1}.2^{-k} = 0$.

Case 2: One instance has $X_n > 0$ and remaining $k-1$ constituent PUFs are affine. Let the non-zero NL be denoted by d. Then, $X_n^k = \frac{1}{2} - 2^{k-1}.2^{-(k-1)}.(\frac{1}{2} - d) = d$. Thus, the NL of the XOR composition is equal to that of the non-linear PUF instance.

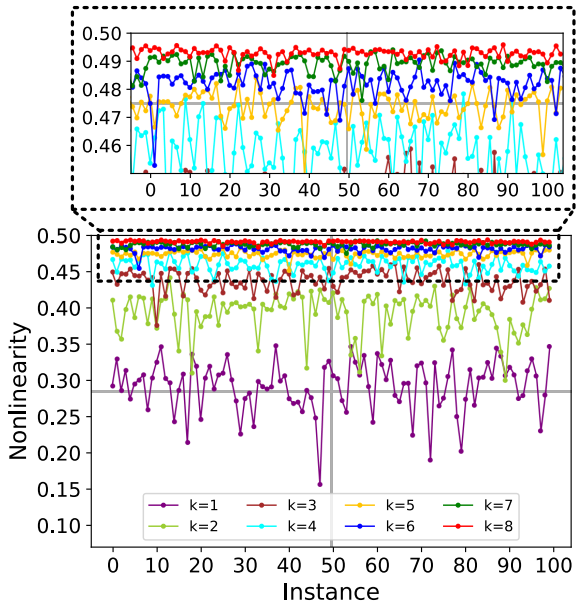


Fig. 14: Non-linearity of individual instances of 20-bit k-XOR BRPUF instances obtained from SAGE ($k = 1, 2, \cdots, 8$)

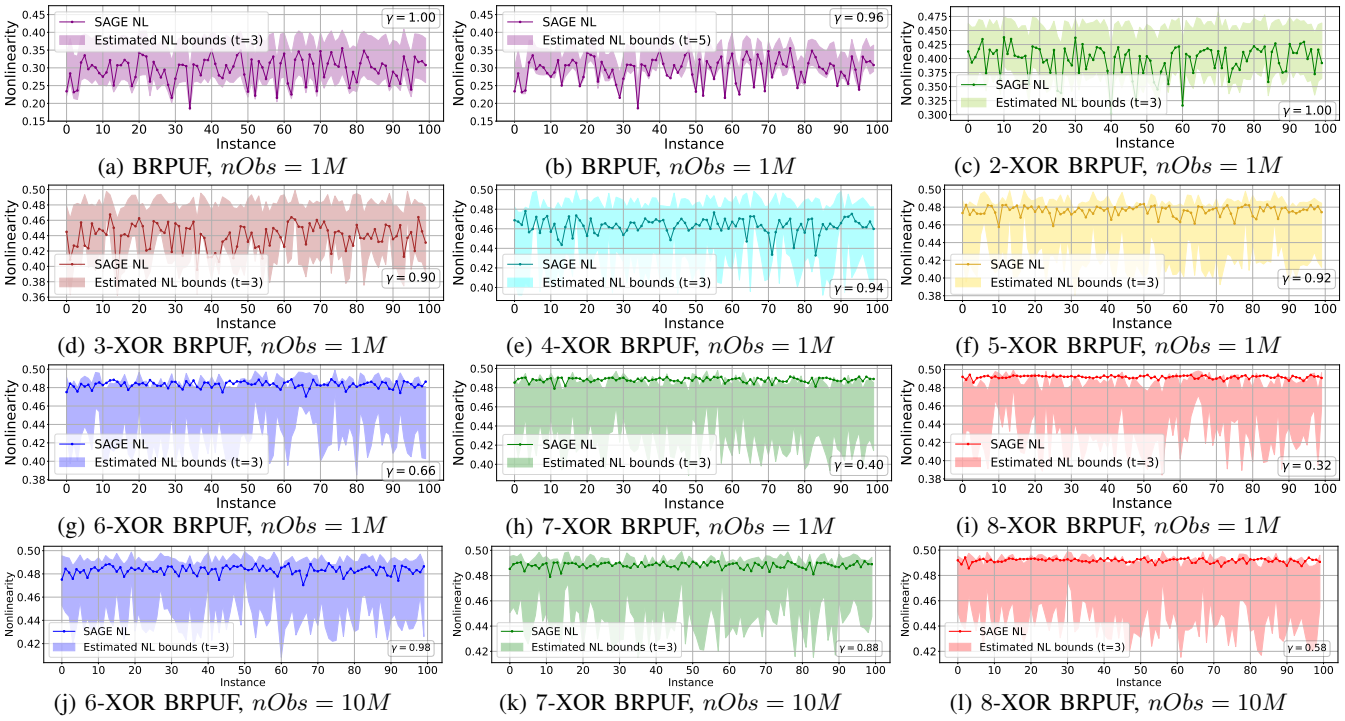
Case 3: Two instances have non-zero NL and the remaining $k-2$ constituent PUFs are affine. Let the non-zero non-linearities be denoted by d_1 and d_2. Then, $X_n^k = \frac{1}{2} - 2^{k-1}.2^{-(k-2)}.(\frac{1}{2} - d_1).(\frac{1}{2} - d_2) = \frac{1}{2} - 2(\frac{1}{2} - d_1).(\frac{1}{2} - d_2) = d_1 + d_2 - 2d_1d_2$. As a result, the NL of the XOR composition is greater than the NL of constituent PUFs.

Since NL is the distance to the closest affine function, we can explain the enhanced NL of XOR composition in terms of distance to affine functions. On XORing the PUF outputs, the distance of the resultant XOR APUF from a given affine function will be at most the sum of distances of the constituent PUFs from that affine function and at least the absolute difference of these distances. To determine the NL, we focus on the distance to the closest affine function. If both the constituent PUFs have the same closest affine function, then the NL can take any value between the absolute difference and sum of their respective distances. On the other hand, in a more probable case, if the constituent PUFs have different closest affine functions, the NL upon XOR will be bounded by the sum of their respective distances. For small k (say $k = 2, 3$), the NL of the XOR composition may even be lesser than the NL of a constituent primitive PUF. One trivial instance of this scenario is when the primitive designs realize the same or similar Boolean function. In this case, the response bits cancel each other and generate a response vector corresponding to an affine function or close to an affine function. This phenomenon is confirmed by the plot for 2-XOR PUF and 2-XOR BRPUF depicted in Fig. 7, 14 respectively. For some 2-XOR APUF instances, the NL value is lower than that of an APUF. However, for higher values of k, the probability of such an occurrence reduces, thereby resulting in higher NL, as depicted in Fig. 7 and 14. Thus, the variation in the NL values is much reduced and close to 0.5. We experimentally validate our hypothesis using APUF and k-XOR APUF instances. First, we demonstrate the efficacy of the heuristic using PUFs with small challenge lengths, for which we can determine the actual NL.

PUFs with Small Challenge lengths: In the first experiment, we apply this mechanism on k-XOR APUF with challenge lengths upto $n = 24$ and $k = \{2, \cdots, 8\}$. We compute the mean estimated NL of XOR APUFs from Eq. 9 using the NL of 20 randomly chosen APUF instances. We compare the estimated value with the average of NL values obtained for k-XOR APUF from the Sagemath tool. Fig 17 illustrates the actual NL, as well as the NL estimated using Piling-up lemma of a 10-bit and a 24-bit XOR APUF for $k = \{1, \cdots, 8\}$. We can observe that on increasing the challenge length, the estimated NL closely approximates the actual NL. Note that this analysis provides us with an estimate of NL of a particular design, rather than for specific instances, which is suitable as the variation in NL for higher XOR values is minimal.

To validate the performance of the proposed heuristic, we perform an instance-specific analysis. Leveraging the PyPUF tool implementation of an XOR APUF, we generate XOR APUF responses along with the contributing arbiter outputs. For PUFs with small challenge lengths, we compute the nonlinearity of each of the constituent APUFs using Sagemath, compute its bias and substitute in Eq.9. We perform this experiment over 100 randomly chosen instances of 24-bit k-XOR APUF, varying $k = 2, \cdots, 8$ and depict the results in Fig. 19. We observe that the nonlinearity estimated by the heuristic provides a close upper bound. As the value of k increases, the difference between the actual and estimated nonlinearity reduces (refer Fig. 19f, 19g). The average of the difference between the actual and estimated nonlinearity reduces from 0.0177 to 0.0056 as k increases from 2 to 8. We repeated the experiment for 200 instances of 20-bit XOR APUF, where this difference reduced from 0.0199 to 0.0084 for $k = 2$ to $k = 8$.

Combination with homomorphicity test: In the next experiment we combine the above-mentioned mechanism with the homomorphicity test. To this end, we substitute the mean of NL bounds obtained from the homomorphicity test for APUF in Eq. 9 with small as well as standard challenge lengths. We vary k in $\{1, \cdots, 8\}$. To validate this estimation, we perform the homomorphicity test on 20 instances of each of the k-XOR APUF constructions and compare the mean of estimated NL bounds with the Piling-up lemma result. The results for k-XOR APUF are presented in Fig. 18. We observe that the outcome of the Piling-up lemma (depicted by blue and red lines) upper bounds the NL interval obtained from the homomorphicity test (depicted by yellow region). Since the Piling-up lemma based test provides an upper bound of the nonlinearity, using γ for estimating the accuracy of the test is not appropriate. Thus, we define a metric β that measures the difference between the nonlinearity bounds obtained from the homomorphicity test for k-XOR compositions and the estimation obtained from Piling-up lemma for each instance. β will take a positive value, as the Piling-up lemma gives an upper bound. Ideally, β should be close to zero. A higher value of β indicates that the homomorphicity test underestimates the nonlinearity of a given instance, thereby implying more



(a) BRPUF, $nObs = 1M$ (b) BRPUF, $nObs = 1M$ (c) 2-XOR BRPUF, $nObs = 1M$

(d) 3-XOR BRPUF, $nObs = 1M$ (e) 4-XOR BRPUF, $nObs = 1M$ (f) 5-XOR BRPUF, $nObs = 1M$

(g) 6-XOR BRPUF, $nObs = 1M$ (h) 7-XOR BRPUF, $nObs = 1M$ (i) 8-XOR BRPUF, $nObs = 1M$

(j) 6-XOR BRPUF, $nObs = 10M$ (k) 7-XOR BRPUF, $nObs = 10M$ (l) 8-XOR BRPUF, $nObs = 10M$

Fig. 15: Non-linearity estimation of individual instances of 20-bit XOR BRPUF

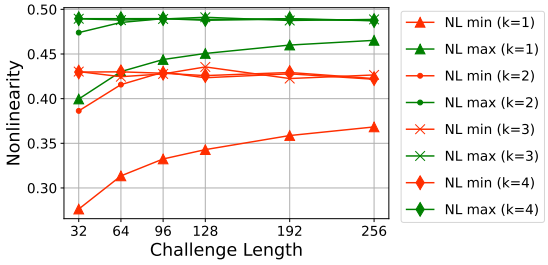
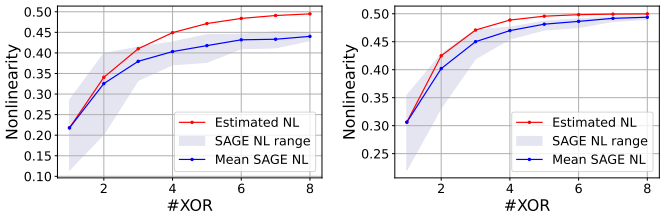


Fig. 16: Non-linearity of n-bit k-XOR APUF estimated using $nObs = 2M$ and $t = 3$



(a) $n = 10$ (b) $n = 24$

Fig. 17: Non-linearity estimation using Piling-up Lemma

observations need to be considered.

V. Discussion and Conclusion

While significant research has been conducted on the ML robustness of a PUF design [49]–[52], [69], [70], little investigation is done on its cryptanalysis [46]–[48]. One of the necessary conditions of a secure cryptographic primitive is high NL. Unlike standard cryptographic primitives such as block ciphers, NL computation of a PUF is computationally infeasible, due to its large challenge length. A block cipher comprises of a set of linear and nonlinear operations. The nonlinearity evaluation of a block cipher is performed by determining the nonlinearity of individual nonlinear components such as Substitution Box (SBox). Since the input to each nonlinear component depends on a subset of input bits, it is feasible to compute the nonlinearity using Walsh transform. On the contrary, the final PUF response depends on all the challenge bits, thereby making it computationally infeasible for PUFs.

This work takes the first step in this direction and proposes a simple, easy-to-standardize and efficient test to estimate the NL of any PUF design using a small fraction of its challenge-response pairs. We illustrate the efficacy of the proposed test using several candidate Strong PUF constructions such as Arbiter PUF, Feed-Forward APUF, Bistable Ring PUF and their XOR compositions with small as well as standard challenge lengths. The experimental results corresponding to PUFs with small challenge lengths provide concrete evidence of the accuracy of the proposed test and insights into the optimal choice of test parameters. We also extend the analysis for designs having standard challenge lengths ($n \geq 64$). Additionally, we propose a mechanism using Piling-up lemma to extend the homomorphicity test for XOR compositions. We validate the upper bound obtained from the mechanism using the NL range estimated by the homomorphicity test.

The test provides insight into the different functionality realized by different instances, as highlighted by the instance-specific analysis. Particularly, we observe that although the average NL of a set of instances for a PUF design is large, there exist some instances that have significantly low NL as

PUF Designs	Design Parameters	Small Challenge Lengths (n) 10	20	24	Standard Challenge Length (n) 32	64	96	128	192	256
APUF	$k=1$	$(10K,3)$ $(1M,7)$	$(10K,3)$ $(1M,5)$	$(10K,3)$ $(1M,5)$	-	-	-	-	-	-
XOR APUF	$k=2$	$(10K,5)$ $(1M,7)$	$(10K,3)$ $(1M,5)$	$(10K,3)$ $(1M,3)$	$(10M,3)$ $\beta_{min}=0.0147$ $\beta_{max}=0.0065$	$(10M,3)$ $\beta_{min}=0.007671$ $\beta_{max}=0.002651$	$(10M,3)$ $\beta_{min}=0.0009$ $\beta_{max}=0.0025$	$(10M,3)$ $\beta_{min}=0.0089$ $\beta_{max}=0.00236$	$(10M,3)$ $\beta_{min}=0.002038$ $\beta_{max}=0.00443$	$(10M,3)$ $\beta_{min}=0.02173$ $\beta_{max}=0.00438$
	$k=3$	$(10K,3)$ $(1M,3)$	$(10K,3)$ $(1M,3)$	$(10K,3)$ $(1M,3)$	$(10M,3)$ $\beta_{min}=0.01997$ $\beta_{max}=0.00477$	$(10M,3)$ $\beta_{min}=0.03300$ $\beta_{max}=0.00608$	$(10M,3)$ $\beta_{min}=0.0404$ $\beta_{max}=0.00672$	$(10M,3)$ $\beta_{min}=0.04538$ $\beta_{max}=0.00718$	$(10M,3)$ $\beta_{min}=0.048713$ $\beta_{max}=0.00729$	$(10M,3)$ $\beta_{min}=0.0502$ $\beta_{max}=0.0072$
	$k=4$	$(10K,3)$ $(1M,3)$	$(10K,3)$ $(1M,3)$	$(10K,3)$ $(1M,3)$	$(10M,3)$ $\beta_{min}=0.050161$ $\beta_{max}=0.007231$	$(10M,3)$ $\beta_{min}=0.051077$ $\beta_{max}=0.007624$	$(10M,3)$ $\beta_{min}=0.05227$ $\beta_{max}=0.00720$	$(10M,3)$ $\beta_{min}=0.05422$ $\beta_{max}=0.00744$	$(10M,3)$ $\beta_{min}=0.05655$ $\beta_{max}=0.00745$	$(10M,3)$ $\beta_{min}=0.05766$ $\beta_{max}=0.00759$
	$k=5$	$(10K,3)$ $(1M,3)$	$(10K,3)$ $(1M,3)$	$(10K,3)$ $(1M,3)$	$(10M,3)$ $\beta_{min}=0.051404$ $\beta_{max}=0.007471$	$(10M,3)$ $\beta_{min}=0.057920$ $\beta_{max}=0.007908$	$(10M,3)$ $\beta_{min}=0.05825$ $\beta_{max}=0.00763$	$(10M,3)$ $\beta_{min}=0.05890$ $\beta_{max}=0.00768$	$(10M,3)$ $\beta_{min}=0.05686$ $\beta_{max}=0.00705$	$(10M,3)$ $\beta_{min}=0.005654$ $\beta_{max}=0.00700$
	$k=6$	$(10K,3)$ $(1M,3)$	$(10K,3)$ $(1M,3)$ $(10M,3)$	$(10K,3)$ $(1M,3)$ $(10M,3)$	$(10M,3)$ $\beta_{min}=0.0569$ $\beta_{max}=0.007673$	$(10M,3)$ $\beta_{min}=0.056414$ $\beta_{max}=0.0071049$	$(10M,3)$ $\beta_{min}=0.05799$ $\beta_{max}=0.00729$	$(10M,3)$ $\beta_{min}=0.05961$ $\beta_{max}=0.00759$	$(10M,3)$ $\beta_{min}=0.05727$ $\beta_{max}=0.00703$	$(10M,3)$ $\beta_{min}=0.06085$ $\beta_{max}=0.00778$
	$k=7$	$(10K,3)$ $(1M,3)$	$(10K,3)$ $(1M,3)$ $(10M,3)$	$(10K,3)$ $(1M,3)$ $(10M,3)$	$(10M,3)$ $\beta_{min}=0.05839$ $\beta_{max}=0.00749$	$(10M,3)$ $\beta_{min}=0.056965$ $\beta_{max}=0.006951$	$(10M,3)$ $\beta_{min}=0.05645$ $\beta_{max}=0.006772$	$(10M,3)$ $\beta_{min}=0.05734$ $\beta_{max}=0.00700$	$(10M,3)$ $\beta_{min}=0.059119$ $\beta_{max}=0.007398$	$(10M,3)$ $\beta_{min}=0.05967$ $\beta_{max}=0.00745$
	$k=8$	$(10K,3)$ $(1M,3)$	$(10K,3)$ $(1M,3)$ $(10M,3)$	$(10K,3)$ $(1M,3)$ $(10M,3)$	$(10M,3)$ $\beta_{min}=0.005691$ $\beta_{max}=0.00702$	$(10M,3)$ $\beta_{min}=0.058364$ $\beta_{max}=0.007221$	$(10M,3)$ $\beta_{min}=0.059156$ $\beta_{max}=0.00730$	$(10M,3)$ $\beta_{min}=0.05605$ $\beta_{max}=0.00678$	$(10M,3)$ $\beta_{min}=0.058725$ $\beta_{max}=0.007294$	$(10M,3)$ $\beta_{min}=0.05722$ $\beta_{max}=0.00700$
FF-APUF	Non overlapping	$(10K,3)$ $(1M,7)$	$(10K,3)$ $(1M,5)$	$(10K,3)$ $(1M,5)$	-	-	-	-	-	-
	Overlapping	$(10K,3)$ $(1M,7)$	$(10K,3)$ $(1M,5)$	$(10K,3)$ $(1M,5)$	-	-	-	-	-	-
BR PUF	$k=1$	$(1M,3)$	$(1M,3)$	$(1M,3)$	$(10M,3)$ $\beta_{min}=0.008471$ $\beta_{max}=0.005713$	$(10M,3)$ $\beta_{min}=0.048988$ $\beta_{max}=0.003349$	$(10M,3)$ $\beta_{min}=0.001230$ $\beta_{max}=0.003219$	$(10M,3)$ $\beta_{min}=0.023311$ $\beta_{max}=0.003439$	$(10M,3)$ $\beta_{min}=0.020910$ $\beta_{max}=0.005849$	$(10M,3)$ $\beta_{min}=0.058891$ $\beta_{max}=0.004345$
XOR BRPUF	$k=2$	$(1M,3)$	$(1M,3)$	$(1M,3)$	$(10M,3)$ $\beta_{min}=0.058910$ $\beta_{max}=0.004589$	$(10M,3)$ $\beta_{min}=0.059081$ $\beta_{max}=0.004594$	$(10M,3)$ $\beta_{min}=0.053012$ $\beta_{max}=0.004491$	$(10M,3)$ $\beta_{min}=0.050993$ $\beta_{max}=0.0040091$	$(10M,3)$ $\beta_{min}=0.057142$ $\beta_{max}=0.003978$	$(10M,3)$ $\beta_{min}=0.048819$ $\beta_{max}=0.003434$
	$k=3$	$(1M,3)$	$(1M,3)$	$(1M,3)$	$(10M,3)$ $\beta_{min}=0.034762$ $\beta_{max}=0.004192$	$(10M,3)$ $\beta_{min}=0.049131$ $\beta_{max}=0.008981$	$(10M,3)$ $\beta_{min}=0.043311$ $\beta_{max}=0.00239$	$(10M,3)$ $\beta_{min}=0.056911$ $\beta_{max}=0.003451$	$(10M,3)$ $\beta_{min}=0.049761$ $\beta_{max}=0.002324$	$(10M,3)$ $\beta_{min}=0.044619$ $\beta_{max}=0.003461$
	$k=4$	$(1M,3)$	$(1M,3)$	$(1M,3)$ $(10M,3)$	$(10M,3)$ $\beta_{min}=0.067591$ $\beta_{max}=0.002884$	$(10M,3)$ $\beta_{min}=0.048871$ $\beta_{max}=0.005881$	$(10M,3)$ $\beta_{min}=0.067819$ $\beta_{max}=0.006781$	$(10M,3)$ $\beta_{min}=0.034571$ $\beta_{max}=0.005381$	$(10M,3)$ $\beta_{min}=0.047381$ $\beta_{max}=0.004785$	$(10M,3)$ $\beta_{min}=0.046817$ $\beta_{max}=0.005289$
	$k=5$	$(1M,3)$	$(1M,3)$	$(1M,3)$ $(10M,3)$	$(10M,3)$ $\beta_{min}=0.047891$ $\beta_{max}=0.005789$	$(10M,3)$ $\beta_{min}=0.058189$ $\beta_{max}=0.004789$	$(10M,3)$ $\beta_{min}=0.047099$ $\beta_{max}=0.005189$	$(10M,3)$ $\beta_{min}=0.055793$ $\beta_{max}=0.007189$	$(10M,3)$ $\beta_{min}=0.047923$ $\beta_{max}=0.008917$	$(10M,3)$ $\beta_{min}=0.046710$ $\beta_{max}=0.005782$
	$k=6$	$(1M,3)$	$(1M,3)$ $(10M,3)$	$(1M,3)$ $(10M,3)$	$(10M,3)$ $\beta_{min}=0.060082$ $\beta_{max}=0.005873$	$(10M,3)$ $\beta_{min}=0.047881$ $\beta_{max}=0.003874$	$(10M,3)$ $\beta_{min}=0.061810$ $\beta_{max}=0.004581$	$(10M,3)$ $\beta_{min}=0.053809$ $\beta_{max}=0.005089$	$(10M,3)$ $\beta_{min}=0.044001$ $\beta_{max}=0.004789$	$(10K,3)$ $\beta_{min}=0.044081$ $\beta_{max}=0.005400$
	$k=7$	$(1M,3)$	$(1M,3)$ $(10M,3)$	$(1M,3)$ $(10M,3)$	$(10M,3)$ $\beta_{min}=0.062081$ $\beta_{max}=0.008901$	$(10M,3)$ $\beta_{min}=0.058010$ $\beta_{max}=0.005380$	$(10M,3)$ $\beta_{min}=0.058213$ $\beta_{max}=0.005099$	$(10M,3)$ $\beta_{min}=0.067921$ $\beta_{max}=0.005801$	$(10M,3)$ $\beta_{min}=0.05377$ $\beta_{max}=0.004980$	$(10M,3)$ $\beta_{min}=0.059830$ $\beta_{max}=0.006499$
	$k=8$	$(1M,3)$ $(10M,3)$	$(1M,3)$ $(10M,3)$	$(1M,3)$ $(10M,3)$	$(10M,3)$ $\beta_{min}=0.058944$ $\beta_{max}=0.008452$	$(10M,3)$ $\beta_{min}=0.057842$ $\beta_{max}=0.006463$	$(10M,3)$ $\beta_{min}=0.057841$ $\beta_{max}=0.007469$	$(10M,3)$ $\beta_{min}=0.059649$ $\beta_{max}=0.007694$	$(10M,3)$ $\beta_{min}=0.06013$ $\beta_{max}=0.006894$	$(10M,3)$ $\beta_{min}=0.059823$ $\beta_{max}=0.008791$

Color																
γ	0-25	26-30	31-35	36-40	41-45	46-50	51-55	56-60	61-65	66-70	71-75	76-80	81-85	86-90	91-95	96-100

TABLE I: Homomorphicity Test parameters for best non-linearity estimation of Arbiter PUF, Bistable Ring PUF and their XOR Compositions computed over 100 randomly chosen instances

depicted in Fig. 9, 14. Thus these instances can be targeted by an attacker for classical cryptanalytic attacks such as linear cryptanalytic attack [56]. Herein, an attacker tries to obtain a linear function that closely approximates the target function.

However, we would like to state that the nonlinearities reported by the homomorphicity test are different to the nonlinearities that guarantee security against ML-attacks. Our homomorphicity test scores and other analyses therefore are useful indicators of the security against cryptanalytic attacks, such as the ones reported in [46]–[48], but *no* direct indicators of a Strong PUF's resilience against ML-attacks. While XOR is a non-linear function with respect to ML-based attacks, it is a linear operator in $\mathbb{F}_2$. In case of an ML-based attack, the XOR operation makes the decision boundary nonlinear in the input space, thereby increasing the modeling complexity. This corroborates with the reduction in ML accuracy for XOR composition of several PUF constructions as the number of XOR inputs (PUF responses combined using XOR) increases [49], [51]. On the other hand, XOR being a linear function in $\mathbb{F}_2$, implies that close linear approximation of PUFs can be used to construct approximations of their XOR compositions. Thus, although XOR strengthens a PUF construction against ML-based attacks, the same cannot be assured for classical cryptanalytic attacks [56]. Also vice versa, the properties that make a PUF resilient against certain cryptanalytic attacks, which we detect in this paper, do not necessarily make it resilient against ML-attacks. Thus exploring the gap between nonlinearity with respect to ML and with respect to mathematical cryptanalysis, and finding complementary metrics to ours that directly assess the security of a PUF against ML-attacks, is an interesting direction for future work. Some first steps in the direction of automatically assessing Strong PUF security against ML-attacks have already been made in [62].

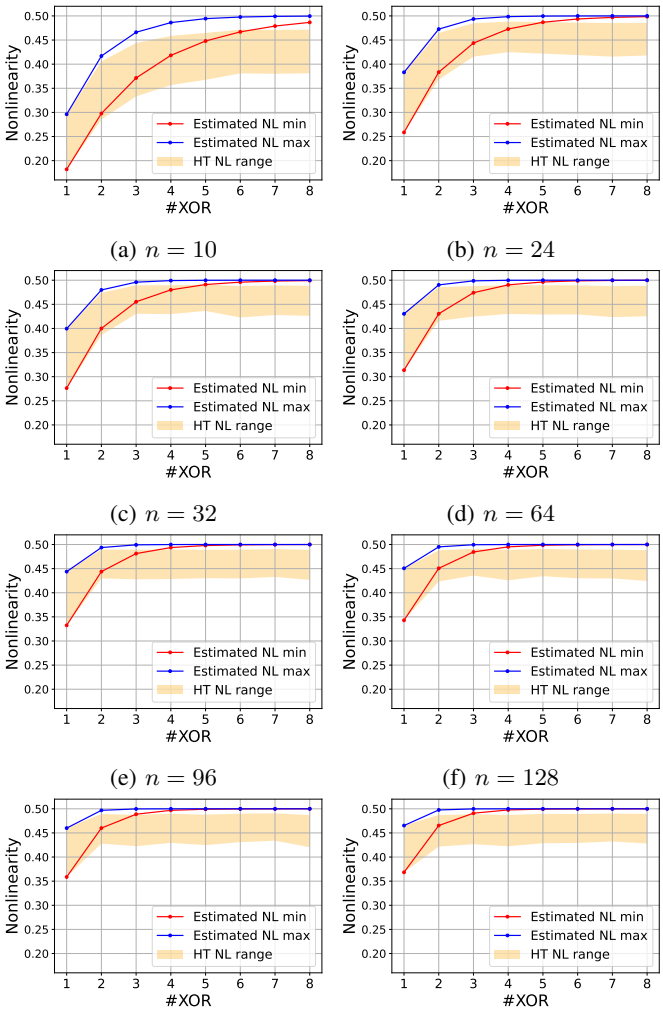
REFERENCES

[1] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[2] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 148–160.

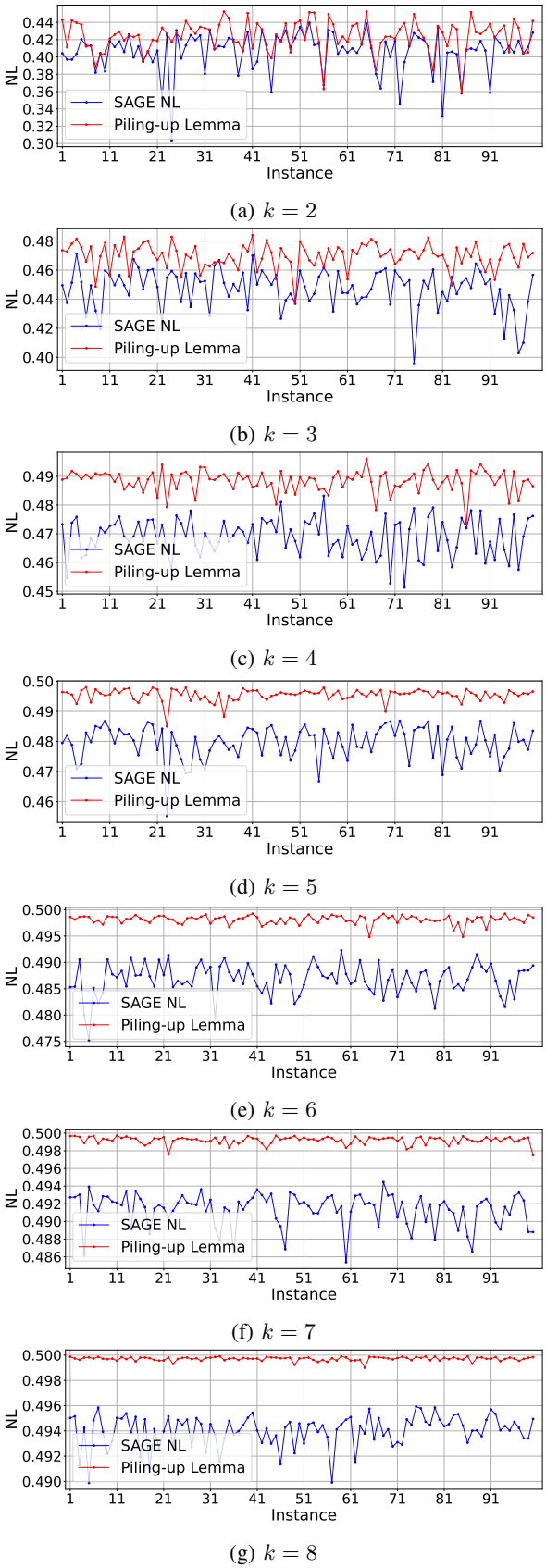
[3] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *2007 44th ACM/IEEE Design Automation Conference*. IEEE, 2007, pp. 9–14.

[4] A.-R. Sadeghi, I. Visconti, and C. Wachsmann, "Puf-enhanced rfid security and privacy," in *Workshop on secure component and system identification (SECSI)*, vol. 110, 2010.



(a) $n = 10$ (b) $n = 24$ (c) $n = 32$ (d) $n = 64$ (e) $n = 96$ (f) $n = 128$ (g) $n = 192$ (h) $n = 256$

Fig. 18: Non-linearity estimation using homomorphicity test and Piling-up Lemma



(a) $k = 2$ (b) $k = 3$ (c) $k = 4$ (d) $k = 5$ (e) $k = 6$ (f) $k = 7$ (g) $k = 8$

Fig. 19: Nonlinearity estimated using Piling up Lemma for 100 instances of 24-bit k-XOR APUF. Note that the y-axis scale is different for each figure.

[5] U. Rührmair, H. Busch, and S. Katzenbeisser, "Strong pufs: models, constructions, and security proofs," in *Towards hardware-intrinsic security*. Springer, 2010, pp. 79–96.

[6] P. Gope, J. Lee, and T. Q. Quek, "Lightweight and practical anonymous authentication protocol for rfid systems using physically unclonable functions," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2831–2843, 2018.

[7] U. Chatterjee, D. Mukhopadhyay, and R. S. Chakraborty, "3paa: A private puf protocol for anonymous authentication," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 756–769, 2020.

[8] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-up sram state as an identifying fingerprint and source of true random numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2008.

[9] C. W. O'donnell, G. E. Suh, and S. Devadas, "Puf-based random number generation," *In MIT CSAIL CSG Technical Memo*, vol. 481, 2004.

[10] U. Rührmair, "Oblivious transfer based on physical unclonable functions," in *International Conference on Trust and Trustworthy Computing*. Springer, 2010, pp. 430–440.

[11] U. Rührmair and M. van Dijk, "Pufs in security protocols: Attack models and security evaluations," in *2013 IEEE symposium on security and privacy*. IEEE, 2013, pp. 286–300.

[12] U. Rührmair, "Physical turing machines and the formalization of physical cryptography," *Cryptology ePrint Archive*, 2011.

[13] M. van Dijk and U. Rührmair, "Protocol attacks on advanced puf protocols and countermeasures," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[14] R. Ostrovsky, A. Scafuro, I. Visconti, and A. Wadia, "Universally composable secure computation with (malicious) physically uncloneable functions," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 702–718.

[15] D. Dachman-Soled, N. Fleischhacker, J. Katz, A. Lysyanskaya, and D. Schröder, "Feasibility and infeasibility of secure computation with malicious pufs," in *Annual Cryptology Conference*. Springer, 2014, pp. 405–420.

[16] U. Rührmair and D. E. Holcomb, "Pufs at a glance," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[17] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, 2014.

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

[19] S. S. Avvaru, Z. Zeng, and K. K. Parhi, "Homogeneous and heterogeneous feed-forward xor physical unclonable functions," *IEEE Trans. on Info. For. & Sec.*, vol. 15, pp. 2485–2498, 2020.

[20] Q. Chen, G. Csaba, X. Ju, S. B. Natarajan, P. Lugli, M. Stutzmann, U. Schlichtmann, and U. Rührmair, "Analog circuits for physical cryptography," in *Proceedings of the 2009 12th International symposium on integrated circuits*. IEEE, 2009, pp. 121–124.

[21] G. Csaba, X. Ju, Z. Ma, Q. Chen, W. Porod, J. Schmidhuber, U. Schlichtmann, P. Lugli, and U. Rührmair, "Application of mismatched cellular nonlinear networks for physical cryptography," in *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*. IEEE, 2010, pp. 1–6.

[22] U. Rührmair, "Simpl systems as a keyless cryptographic and security primitive," in *Cryptography and Security: From Theory to Applications*. Springer, 2012, pp. 329–354.

[23] 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*, vol. 10, no. 3, pp. 489–498, 2010.

[24] P. Lugli, A. Mahmoud, G. Csaba, M. Algasinger, M. Stutzmann, and U. Rührmair, "Physical unclonable functions based on crossbar arrays for cryptographic applications," *International journal of circuit theory and applications*, vol. 41, no. 6, pp. 619–633, 2013.

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

[26] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, and U. Rührmair, "The bistable ring puf: A new architecture for strong physical unclonable functions," in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, 2011, pp. 134–141.

[27] M. Kalyanaraman and M. Orshansky, "Novel strong puf based on nonlinearity of mosfet subthreshold operation," in *2013 IEEE international symposium on hardware-oriented security and trust (HOST)*. IEEE, 2013, pp. 13–18.

[28] Y. Wang, X. Xi, and M. Orshansky, "Lattice puf: A strong physical unclonable function provably secure against machine learning attacks," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 273–283.

[29] A. Vijayakumar and S. Kundu, "A novel modeling attack resistant puf design based on non-linear voltage transfer characteristics," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 653–658.

[30] D. P. Sahoo, D. Mukhopadhyay, R. S. Chakraborty, and P. H. Nguyen, "A multiplexer-based arbiter puf composition with enhanced reliability and security," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 403–417, 2017.

[31] F. Tehranipoor, N. Karimian, K. Xiao, and J. Chandy, "Dram based intrinsic physical unclonable functions for system level security," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, 2015, pp. 15–20.

[32] J. S. Kim, M. Patel, H. Hassan, and O. Mutlu, "The dram latency puf: Quickly evaluating physical unclonable functions by exploiting the latency-reliability tradeoff in modern commodity dram devices," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 194–207.

[33] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "The butterfly puf protecting ip on every fpga," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 2008, pp. 67–70.

[34] P. Simons, E. van der Sluis, and V. van der Leest, "Buskeeper pufs, a promising alternative to d flip-flop pufs," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, 2012, pp. 7–12.

[35] K. Lofstrom, W. R. Daasch, and D. Taylor, "Ic identification circuit using device mismatch," in *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No. 00CH37056)*. IEEE, 2000, pp. 372–373.

[36] U. Rührmair, C. Jaeger, C. Hilgers, M. Algasinger, G. Csaba, and M. Stutzmann, "Security applications of diodes with unique current-voltage characteristics," in *International Conference on Financial Cryptography and Data Security*. Springer, 2010, pp. 328–335.

[37] A. Shamsoshoara, A. Korenda, F. Afghah, and S. Zeadally, "A survey on physical unclonable function (puf)-based security solutions for internet of things," *Computer Networks*, vol. 183, p. 107593, 2020.

[38] S. Yu and Y. Park, "A robust authentication protocol for wireless medical sensor networks using blockchain and physically unclonable functions," *IEEE Internet of Things Journal*, 2022.

[39] G. Bansal, N. Naren, V. Chamola, B. Sikdar, N. Kumar, and M. Guizani, "Lightweight mutual authentication protocol for v2g using physical unclonable function," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7234–7246, 2020.

[40] T. A. Idriss, H. A. Idriss, and M. A. Bayoumi, "A lightweight puf-based authentication protocol using secret pattern recognition for constrained iot devices," *IEEE Access*, vol. 9, pp. 80 546–80 558, 2021.

[41] U. Chatterjee, V. Govindan, R. Sadhukhan, D. Mukhopadhyay, R. S. Chakraborty, D. Mahata, and M. M. Prabhu, "Building PUF based authentication and key exchange protocol for iot without explicit crps in verifier database," *IEEE Trans. Dependable Secur. Comput.*, vol. 16, no. 3, pp. 424–437, 2019. [Online]. Available: https://doi.org/10.1109/TDSC.2018.2832201

[42] C. Brzuska, M. Fischlin, H. Schröder, and S. Katzenbeisser, "Physically uncloneable functions in the universal composition framework," in *Annual Cryptology Conference*. Springer, 2011, pp. 51–70.

[43] U. Rührmair and M. van Dijk, "On the practical use of physical unclonable functions in oblivious transfer and bit commitment protocols," *Journal of Cryptographic Engineering*, vol. 3, no. 1, pp. 17–28, 2013.

[44] S. Khaleghi and W. Rao, "Hardware obfuscation using strong pufs," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2018, pp. 321–326.

[45] H. Kareem and D. Dunaev, "Physical unclonable functions based hardware obfuscation techniques: A state of the art," in *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2021, pp. 1–6.

[46] D. P. Sahoo, P. H. Nguyen, D. Mukhopadhyay, and R. S. Chakraborty, "A case of lightweight puf constructions: Cryptanalysis and machine learning attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1334–1343, 2015.

[47] L. Kraleva, M. Mahzoun, R. Posteuca, D. Toprakhisar, T. Ashur, and I. Verbauwhede, "Cryptanalysis of strong physically unclonable functions," *IEEE Open Journal of the Solid-State Circuits Society*, 2022.

[48] P. H. Nguyen and D. P. Sahoo, "An efficient and scalable modeling attack on lightweight secure physically unclonable function," *Cryptology ePrint Archive*, 2016.

[49] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 237–249.

[50] N. Wisiol, C. Mühl, N. Pirnay, P. H. Nguyen, M. Margraf, J.-P. Seifert, M. van Dijk, and U. Rührmair, "Splitting the interpose puf: A novel modeling attack strategy," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 97–120, 2020.

[51] N. Wisiol, B. Thapaliya, K. T. Mursi, J.-P. Seifert, and Y. Zhuang, "Neural network modeling attacks on arbiter-puf-based designs," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2719–2731, 2022.

[52] M. Khalafalla and C. Gebotys, "Pufs deep attacks: Enhanced modeling attacks using deep learning techniques to break the security of double arbiter pufs," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 204–209.

[53] G. T. Becker, "The gap between promise and reality: On the insecurity of xor arbiter pufs," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 535–555.

[54] D. Chatterjee, U. Chatterjee, D. Mukhopadhyay, and A. Hazra, "Sacred: An attack framework on sac resistant delay-pufs leveraging bias and reliability factors," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 85–90.

[55] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "Puf modeling attacks on simulated and silicon data," *IEEE transactions on information forensics and security*, vol. 8, no. 11, pp. 1876–1891, 2013.

[56] M. Matsui, "Linear cryptanalysis method for des cipher," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1993, pp. 386–397.

[57] F. Ganji, D. Forte, and J.-P. Seifert, "Pufmeter a property testing tool for assessing the robustness of physically unclonable functions to machine learning attacks," *IEEE Access*, vol. 7, pp. 122 513–122 521, 2019.

[58] D. Chatterjee, D. Mukhopadhyay, and A. Hazra, "PUF-G: A CAD framework for automated assessment of provable learnability from formal PUF representations," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[59] D. Chatterjee, A. Hazra, and D. Mukhopadhyay, "Formal analysis of puf instances leveraging correlation-spectra in boolean functions," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2019, pp. 142–158.

[60] A. Roy, D. Roy, and S. Maitra, "How Do the Arbiter PUFs Sample the Boolean Function Class?" in *International Conference on Selected Areas in Cryptography*. Springer, 2022, pp. 111–130.

[61] N. Linial, Y. Mansour, and N. Nisan, "Constant depth circuits, fourier transform, and learnability," *Journal of the ACM (JACM)*, vol. 40, no. 3, pp. 607–620, 1993.

[62] F. Kappelhoff, R. Rasche, D. Mukhopadhyay, and U. Rührmair, "Strong puf security metrics: Response sensitivity to small challenge perturbations," in *2022 23rd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2022, pp. 1–10.

[63] X.-M. Zhang and Y. Zheng, "The nonhomomorphicity of boolean functions," in *International Workshop on Selected Areas in Cryptography*. Springer, 1998, pp. 280–295.

[64] A. Sălăgean and P. Stănică, "Improving bounds on probabilistic affine tests to estimate the nonlinearity of boolean functions," *Cryptography and Communications*, vol. 14, no. 2, pp. 459–481, 2022.

[65] R. O'Donnell, *Analysis of boolean functions*. Cambridge University Press, 2014.

[66] X. Xu, U. Rührmair, D. E. Holcomb, and W. Burleson, "Security evaluation and enhancement of bistable ring pufs," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*. Springer, 2015, pp. 3–16.

[67] N. Wisiol, C. Gräbnitz, C. Mühl, B. Zengin, T. Soroceanu, N. Pirnay, K. T. Mursi, and A. Baliuka, "pypuf: Cryptanalysis of Physically Unclonable Functions," 2021. [Online]. Available: https://doi.org/10.5281/zenodo.3901410

[68] The Sage Developers, *SageMath, the Sage Mathematics Software System (Version x.y.z)*, 2022, `https://www.sagemath.org`.

[69] M. S. Alkatheiri and Y. Zhuang, "Towards fast and accurate machine learning attacks of feed-forward arbiter pufs," in *2017 IEEE Conference on Dependable and Secure Computing*. IEEE, 2017, pp. 181–187.

[70] M. Khalafalla, M. A. Elmohr, and C. Gebotys, "Going deep: Using deep learning techniques with simplified mathematical models against xor br and tbr pufs (attacks and countermeasures)," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 80–90.