Architectural Bias: a Novel Statistical Metric to Evaluate Arbiter PUF Variants

Durga Prasad Sahoo*, Phuong Ha Nguyen†, Rajat Subhra Chakraborty‡ and Debdeep Mukhopadhyay§
SEAL/CSE, Indian Institute of Technology Kharagpur, INDIA-721302
Email: {dpsahoo*,rschakraborty‡,debdeep§}@cse.iitkgp.ernet.in & phuongha.ntu@gmail.com†

***Abstract*—This paper introduces the notion of *Architectural Bias*, which can be used to measure the influence of the architecture of Arbiter Physically Unclonable Functions (APUFs) on the quality of its outputs. A PUF design with less architectural bias is better than one which has more architectural bias. Architectural bias is the bias in the challenge-response behavior of a PUF due to the architectural features of the design itself, independent of the implementation platform, technology node and external factors. This bias is different from the bias observed in the APUF outputs when implemented on Field Programmable Gate Array (FPGA) platform. This platform induced bias is called as *Implementation Bias*. To overcome the effect of implementation bias in classic APUF, *Programmable Delay Line APUF* (PAPUF) and *Double APUF* (DAPUF) have been proposed as alternatives for APUF on FPGA platforms. In this work, we provide a comparative analysis of the architectures of APUF and its two design variants based on the derived linear additive delay models. Subsequently, these designs are evaluated with the architectural bias to quantify the number of good (i.e. usable) PUF instances that it can generate. We also develop a scheme to perform instance-level comparison of a pair of randomly selected PUF instances of two different PUF designs. In addition, we study the impacts of architectural bias on PUF performance metrics namely *uniformity*, *uniqueness* and *reliability*. We validate our theoretical findings with simulation and FPGA-based implementation results. Experimental results reveal that the classic APUF has the least architectural bias, followed by the DAPUF and the PAPUF, respectively.**

***Index Terms*—Architectural bias, arbiter PUF (APUF), double APUF (DAPUF), physically unclonable function (PUF), programmable delay line (PDL), PDL based APUF (PAPUF).**

I. Introduction

A physically unclonable function (PUF) is a physical random function, and its input-output mapping is determined by static, unpredictable and uncontrollable instance-specific manufacturing-induced variations. PUFs have been proposed as important building blocks in various secure applications since first introduced in [1], [2], e.g. device identification and authentication [3], binding software to hardware platforms [4], secure storage of cryptographic secrets [5], secure protocol design [6] and "keyless" cipher design [7].

The Arbiter PUF (APUF) [2] is one of the most widely studied delay variation based *strong PUF* architectures (a "strong PUF" is one in which the challenge space is large enough to make trivial attacks infeasible). Principal features of the APUF are its regular and lightweight design, and easy to analyze structure. However, inventors of the APUF design themselves showed that this design can be mathematically modelled by machine learning techniques, because it follows a *Linear Additive Delay Model* [2]. An adversary can build an accurate mathematical model of a given APUF instance from a small subset of its challenge-response database, using machine learning approaches such as *Support Vector Machines* (SVM), *Logistic Regression* (LR) and Evolution Strategy (ES). This type of mathematical cloning is known as *Model Building Attack* [8]–[12], and enables an adversary to predict the response for a previously unseen challenge with a high probability of success. In spite of this known vulnerability, APUFs have been used as building blocks in the design of various delay PUF variants, namely *Exclusive-OR PUF* (XOR PUF), *Lightweight Secure PUF* (LSPUF) [13], and *Composite PUF* [14], [15], mainly because of its relatively low hardware resource footprint. In these designs, one or more APUF instances are usually combined with additional non-linear layer(s) to improve the modeling robustness, although later it is observed that these designs are also vulnerable to modeling and side-channel attacks due to the flaw in composition schemes [8]–[11], [16], [17]. Along with robustness to modeling attacks, the suitability of a given PUF variant is judged based on the evaluation of a few statistical metrics, the most important among which are: *uniformity*, *uniqueness* and *reliability* [18], [19]. The versatile applications of APUF as a fundamental PUF design block motivate us to revisit this design and its variants proposed for the FPGA platform.

Implementation of APUF on FPGA devices is a challenging problem due to predefined logic array layout, and relative lack of designer control on routing offered by existing FPGA CAD tools. Consequently, implementation induced bias creeps into the APUFs implemented on FPGAs, and results in APUF implementation with relatively poor statistical properties. To overcome the shortcomings, researchers came up with two APUF design variants specifically targeting FPGA devices: *Programmable Delay Line* (PDL) based APUF (PAPUF) [20] and *Double Arbiter PUF* (DAPUF) [21], [22]. In PAPUF, switching stages exploit non-swapping paths to reduce the design induced bias, in place of swapping paths based switches in the classic APUF. The DAPUF on the other hand, is a more elegant construction which does not modify the basic switch structure, but exploits two separate APUFs to eliminate the effect of possible implementation bias.

Since their introduction and in spite of their relatively frequent use, to the best of our knowledge, PAPUF and DAPUF have not been modelled analytically to elucidate their strengths and weaknesses. A statistical analysis methodology similar to the one employed by us in this paper was used in [23] to analyze delay PUFs. However, that work did not

consider PAPUF and DAPUF, which are generally claimed to be more suitable APUF variants for FPGAs. Furthermore, an exploration of the architectures of these variants enable us to measure the effect of the architecture on the bias in PUF behavior. We call this architecture induced bias as "Architectural Bias" of the PUF design.

Our major contributions in this work are summarized as follows:

1) We develop analytical delay models for the DAPUF and PAPUF circuits.
2) We introduce the concept of *architectural bias*, and quantitatively estimate it from the analytical delay models derived.
3) We apply the concept of architectural bias to evaluate the goodness of a PUF design from the context of the fraction of "good" (i.e. usable) PUF instances that can be expected in a given population. As case studies, we evaluate the APUF, PAPUF and DAPUF designs using the proposed evaluation scheme. We show that for bias tolerance parameter $\epsilon = 0.1$, 99% of APUF instances are good, but this quantity is 33% for DAPUF and 20% for PAPUF.
4) We also provide a scheme to perform the instance-level comparison of PUF designs. Specifically, the proposed scheme allows one to decide which instance is better among a given pair of randomly selected PUF instances of two different PUF designs. In addition, we extend this instance-level analysis to design-level, so that we can compare the PUF designs also.
5) We discuss the impact of architectural bias on PUF quality metrics: uniformity, uniqueness, and reliability.
6) We realize the APUF, PAPUF and DAPUF designs by using *Matlab* based simulation and FPGA implementations, to validate the notion of architectural bias. Our analysis predicts the classic APUF architecture to have superior uniformity and uniqueness to PAPUF and DAPUF architectures, but inferior reliability. We also show that DAPUF is comparatively better than the PAPUF with respect to uniformity, and uniqueness, with lesser inherent architectural bias. These theoretical trends are supported by our simulation results.
7) Finally, we provide experimental evaluation of the three PUF variants on *Xilinx* FPGA, which establishes that the DAPUF is a better APUF variant than PAPUF for FPGAs.

The rest of the paper is organized as follows. In Section II, we introduce a notation system that will be used throughout the paper, and we briefly discuss about the important PUF quality metrics. The notion of bias in PUF challenge-response behavior is explained in Section III. In Section IV, we revisit the analytical delay model of the APUF, and develop those for the PAPUF and DAPUF respectively. Definition of architectural bias, and its application as a metric to evaluate the APUF, PAPUF and DAPUF designs is discussed in Section V. An instance-level comparison scheme of PUF designs is developed in Section VI. In Section VII, we discuss about the impact of architectural bias on uniqueness and reliability properties. Experimental results are provided in Section VIII. Concluding comments and future directions of research are provided in Section IX.

II. PRELIMINARIES

A. Notations

The following notation system will be used throughout the paper. A vector is represented by lowercase letter in bold font, e.g $\mathbf{a}$. A vector with m-components is represented as: $\mathbf{a} = (\mathbf{a}[0], \ldots, \mathbf{a}[i], \ldots, \mathbf{a}[m-1])$, where $\mathbf{a}[i]$ is the $(i+1)$th component of the vector. We use $\mathbf{a}[i:j]$ as a sub-vector $(\mathbf{a}[i], \ldots, \mathbf{a}[j])$. The normal lowercase letter denotes a scalar, e.g. n. A set is represented by the calligraphic font, e.g. $\mathcal{D}$, and its cardinality is denoted as $|\mathcal{D}|$. We denote a random variable by using a upper-case letter, e.g. X; $Pr(X = x)$ is used to denote the probability of the event $X = x$, and μ_{X}, σ^2_{X} are used to denote mean and variance of the random variable X, respectively. Random variable X following a Gaussian probability distribution function is denoted as $X \sim \mathcal{N}(\mu, \sigma^2)$. $\phi_{\mu,\sigma^2}()$ and $\Phi_{\mu,\sigma^2}()$ represent the probability density function (PDF) and cumulative distribution function (CDF) of Gaussian random variable, respectively; $\phi()$ and $\Phi()$ represent the PDF and CDF of standard normal random variable, respectively.

The Hamming weight of a binary vector $\mathbf{a}$ is denoted by $\mathrm{HW}(\mathbf{a})$ and the Hamming distance of two vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted by $\mathrm{HD}(\mathbf{a}, \mathbf{b})$.

In the analytical delay models for the PUF variants, the input to the PUF ("challenge") follows a bipolar encoding, i.e. logic-0 is encoded as +1 and logic-1 is encoded as -1.

B. PUF Quality Metrics

Quality evaluation of a given PUF variant is typically performed based on the collected challenge-response pairs (CRPs) of several PUF instances of the same type on a specific platform. The three most popular PUF quality metrics are [18], [19]: uniformity, uniqueness, and reliability. In this context, we consider only the PUFs which generate 1-bit response for a given challenge, and n-bit response string is generated by evaluating the PUF with respect to n different challenges. Next we provide a brief description of these quality metrics.

1) Reliability (S): The challenge-response behavior of PUFs are affected by time-varying dynamic noise that are generated due to mainly temperature and supply voltage variations. In addition, variation due to the aging of PUF circuit modifies the circuit behavior permanently. Together these factors imply that the behavior of a given PUF instance is not stable. Typically, golden challenge-response pairs (CRPs) are defined by employing *Majority Voting* on the CRPs obtained from m different evaluations of PUF at normal operating condition. Let $\mathbf{r}_{i,t}$ be a n-bit response of ith PUF instance at time $t = 1, \ldots, m$, $i = 1, \ldots, k$. Golden response bits are formally defined as follows:

$$\mathbf{r}_i[j] = \left\lfloor 0.5 + \frac{1}{m} \sum_{t=1}^{m} \mathbf{r}_{i,t}[j] \right\rfloor, \tag{1}$$

where $j = 1, \ldots, n$.

Reliability of ith PUF instance is defined as the fraction of response bits matches with corresponding golden response bits.

$$S_i = 1 - \frac{1}{m}\sum_{t=1}^{m}\frac{\mathrm{HD}(\mathbf{r}_{i,t}, \mathbf{r}_i)}{n} \tag{2}$$

In case of an ideal PUF, $S_i = 1$, i.e., PUF challenge-response is perfectly reliable or stable. When expressed as a percentage, the ideal value of reliability is 100%.

2) Uniformity (B): Since every PUF instance can be treated as a physical random function, ideally it should be balanced in the number of 0's and 1's in a generated response bitstring. This property is termed as *Uniformity*. Uniformity of a PUF instance is computed based on its golden responses as defined in Eq. (1). Typically, uniformity (in percentage) of ith PUF instance is computed as the fraction of 1's in the n-bit response:

$$B = \frac{\mathrm{HW}(\mathbf{r}_i)}{n} \times 100\% \tag{3}$$

Ideal value of uniformity metric is $B = 50\%$.

3) Uniqueness (U): Uniqueness is estimated by the average inter Hamming distance among the responses of a set of PUF instances of the same type. This metric is used to estimate to what extent the input-output behavior of instances of a given type of PUF identify each instance uniquely. The statistical estimation of this metric needs CRPs of multiple PUF instances, provided that the same set of challenges is applied to all the PUF instances. Let $\mathbf{r}_i$ and $\mathbf{r}_j$ be the n-bit responses of ith and jth PUF instances, respectively. Then, the uniqueness metric (in percentage) is computed by:

$$U = \frac{2}{k(k-1)}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k}\frac{\mathrm{HD}(\mathbf{r}_i, \mathbf{r}_j)}{n} \times 100\%, \tag{4}$$

where k is the number of PUF instances. Ideal value of uniqueness is $U = 50\%$.

III. Notion of Bias in PUF Challenge-Response Behavior

In case of delay PUFs, the response to a given challenge $\mathbf{c}$ is determined by the delay difference $\Delta_{\mathbf{c}}$ as:

$$r = \begin{cases} 1 & \text{if } \Delta_{\mathbf{c}} < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

It is desirable that an ideal delay PUF should generate responses where 0's and 1's are equiprobable, i.e.,

$$\begin{aligned} B = Pr(r=1) &= Pr(\Delta \le 0) \\ &= \frac{1}{\sigma_\Delta\sqrt{2\pi}}\int_{-\infty}^{0} e^{-\frac{(u-\mu_\Delta)^2}{2\sigma_\Delta^2}}\,\mathrm{d}u \\ &= \Phi_{\mu_\Delta,\sigma_\Delta^2}(0) = \Phi\left(\frac{-\mu_\Delta}{\sigma_\Delta}\right), \end{aligned} \tag{6}$$

where $\Delta \sim \mathcal{N}(\mu_\Delta, \sigma_\Delta^2)$ is a Gaussian random variable representing the delay differences of a delay PUF instance. Equation (6) is another way of stating that the ideal value of PUF uniformity is $B = \Phi(0) = 0.5$ when $(\frac{-\mu_\Delta}{\sigma_\Delta}) = 0$. If a PUF design fails to achieve this property, i.e., $(\frac{-\mu_\Delta}{\sigma_\Delta}) \neq 0$, then its challenge-response behavior exhibits a *bias*. The bias value can be estimated as:

$$b = |\Phi(0) - B| = |0.5 - B|\,. \tag{7}$$

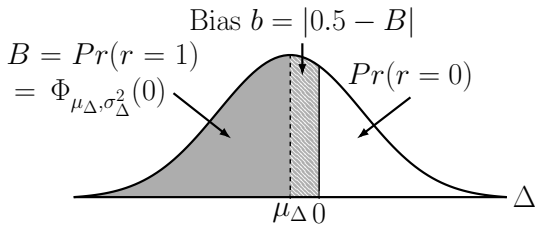


Fig. 1: An example delineating bias in the behavior of a delay PUF instance. In this case, response generated by a PUF instance is biased towards '1'.

Figure 1 shows the behavior of a practical delay PUF instance where $\mu_\Delta < 0$, and it results a bias in the behavior of PUF instance towards '1', since the response is defined by Eq. (5).

Two key reasons for this bias can be:

1) **Imperfect Implementation.** The most important requirement while designing an APUF is symmetric placement and routing of its delay stages, such that the nominal delay difference between any two delay paths is zero. In other words, the only factor affecting delay differences should be process-variation induced device-level random variations. This requirement is almost infeasible to achieve in case of the FPGA platform due to predefined layout. It results in a bias due to non-ideal implementation of the classic APUF on FPGA. We term this bias as *implementation bias*.
2) **Architectural Weakness.** Some PUF designs might exhibit bias in their performance, even when their ideal (bias-free) implementation is feasible. This bias is due to the inherent architectural weakness of PUF design, and we term this bias as *Architectural Bias*.

In this work, we mainly focus on the architectural bias of a PUF design, and as case studies, we discuss the notion of architectural bias in the APUF, PAPUF, and DAPUF designs based on their linear additive delay models. Next we revisit the delay model of the classic APUF, and derive the delay models for the PAPUF and the DAPUF.

IV. Linear Additive Delay Models of APUF, PAPUF and DAPUF

A. Delay Model of APUF

Figure 2 shows the classic APUF design, with a cascade of path-swapping switches and an *arbiter* at the end. Each path-swapping switch can be implemented using a pair of 2:1 multiplexors. Starting from a common point, a voltage impulse propagates along two symmetrical paths defined by the applied control bits to the switches. The individual stage delays are randomly determined by device-level process variation effects, and are unpredictable. The accumulated delay difference along

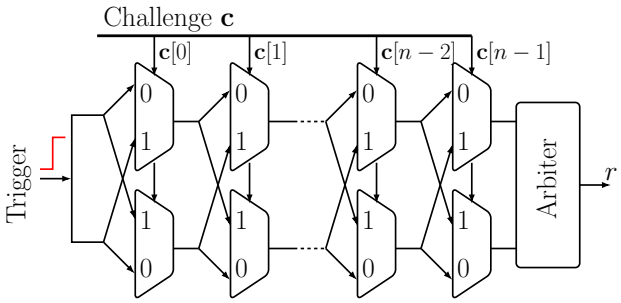


Fig. 2: A n-stage classic Arbiter PUF with challenge $\mathbf{c} \in \{0,1\}^n$.

the two paths determines the output of the arbiter circuit, which is usually an edge-triggered D flip-flop (DFF) or a SR latch. The control bit string is considered as the applied *challenge*, and the output of arbiter is considered as *response*.

Now, we briefly discuss about the linear additive delay model of AUF as derived in [2]. Let p_i, r_i, s_i and q_i be the four delay components of the ith switching stage of APUF as shown in Fig. 3a. Let us denote $\delta_t(i)$ and $\delta_b(i)$ as the timing delays from the starting point to the end of the ith stage of the top and bottom paths, respectively. Then,

$$\begin{aligned}\delta_t(i+1) &= \frac{1+\mathbf{c}[i+1]}{2}(p_{i+1}+\delta_t(i)) \\ &\quad + \frac{1-\mathbf{c}[i+1]}{2}(s_{i+1}+\delta_b(i)) \\ \delta_b(i+1) &= \frac{1+\mathbf{c}[i+1]}{2}(q_{i+1}+\delta_b(i)) \\ &\quad + \frac{1-\mathbf{c}[i+1]}{2}(r_{i+1}+\delta_t(i)),\end{aligned}$$

where each challenge bit $\mathbf{c}[i] \in \{+1,-1\}$ is bipolar-encoded. Let $\Delta(i+1) = \delta_t(i+1) - \delta_b(i+1)$. Then:

$$\Delta(i+1) = \Delta(i)\mathbf{c}[i+1] + \alpha_{i+1}\mathbf{c}[i+1] + \beta_{i+1}, \quad (8)$$

where

$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2} \quad \text{and} \quad \beta_i = \frac{p_i - q_i - r_i + s_i}{2}.$$

Setting $\Delta(-1) = 0$ and using Eq. (8), iteratively we can define the delay difference of top and bottom paths of APUF as:

$$\Delta(n-1) = \mathbf{w}[0]\mathbf{\Phi}[0] + \mathbf{w}[1]\mathbf{\Phi}[1] + \cdots + \mathbf{w}[n]\mathbf{\Phi}[n], \quad (9)$$

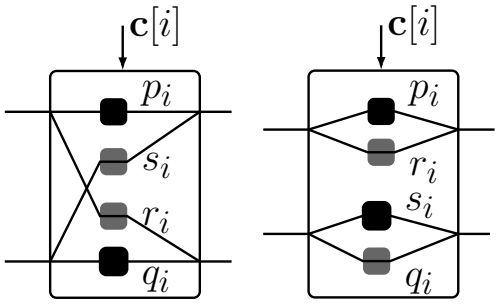
where the feature vector $\mathbf{\Phi}$ is defined as a parity of challenge bits:

$$\mathbf{\Phi}[i] = \begin{cases} 1 & \text{if } i = n, \\ \prod_{j=i}^{n-1} \mathbf{c}[j] & \text{if } i = 0,\ldots,n-1. \end{cases} \quad (10)$$

The vector $\mathbf{w}$ is defined as follows:

$$\mathbf{w}[i] = \begin{cases} \alpha_0 & \text{if } i = 0, \\ \alpha_i + \beta_{i-1} & \text{if } i = 1,\ldots,n-1, \\ \beta_{n-1}, & \text{if } i = n. \end{cases} \quad (11)$$

Hence, $\Delta(n-1) = \mathbf{w} \cdot \mathbf{\Phi}^{\mathsf{T}}$ represents a linear additive delay model of an APUF.



(a) APUF stage (b) PAPUF stage

Fig. 3: Comparison of delay stages in classic APUF and PAPUF. Depending on the challenge bit $\mathbf{c}[i]$ a pair of delay elements of similar type (similar shade in figure) is selected to form a pair of identical paths.

B. Delay Model for PAPUF

Architecturally, the classic APUF and the PAPUF are very similar. The principal difference between APUF and PAPUF on FPGAs lies in the design of the switching elements. Fig. 3 shows the structural difference between the switching elements of APUF and PAPUF. In contrast to APUFs which use path-swapping switches, PAPUFs use non-swapping path based switches to reduce the implementation bias. For a given challenge, upper MUXes of switches form a path while lower MUXes form another path. These two paths are independent, and hence symmetric routing is easier to maintain for them using the *hard macro* option in Xilinx CAD tools. Unlike APUF, the top path and bottom path in PAPUF do not cross each other along the entire chain of switches, and this is a major implementation advantage of PAPUF on FPGA devices.

Let $\Delta(n-1)$ be the delay difference of top and bottom paths after the final switch for a given n-bit challenge $\mathbf{c}$, where $\mathbf{c}$ follows a bipolar encoding. The sign of $\Delta(n-1)$ determines the response for a given challenge, as in the case of the classic APUF.

Next we develop a linear additive delay model for PAPUF. Let p_{i+1} and r_{i+1} be two delay components of the top MUX in the $(i+1)$th stage, and s_{i+1} and q_{i+1} be the other two delay components of the bottom MUX in the $(i+1)$th stage. Let $\delta_t(i)$ and $\delta_b(i)$ be the propagation delays of the trigger signal along the top and bottom paths to the input of the $(i+1)$th stage of PAPUF, respectively. For a given challenge $\mathbf{c} \in \{+1,-1\}$, propagation delay of the trigger signal at the output of the $(i+1)$th stage is given by:

$$\begin{aligned}\delta_t(i+1) &= \frac{1+\mathbf{c}[i+1]}{2}(\delta_t(i)+p_{i+1}) \\ &\quad + \frac{1-\mathbf{c}[i+1]}{2}(\delta_t(i)+r_{i+1}) \\ \delta_b(i+1) &= \frac{1+\mathbf{c}[i+1]}{2}(\delta_b(i)+s_{i+1}) \\ &\quad + \frac{1-\mathbf{c}[i+1]}{2}(\delta_b(i)+q_{i+1})\end{aligned}$$

Then delay difference between top and bottom paths can be

estimated as follows:

$$\begin{aligned}\Delta(i+1) &= \delta_t(i+1) - \delta_b(i+1) \\ &= \frac{1+\mathbf{c}[i+1]}{2}(\Delta(i) + p_{i+1} - s_{i+1}) \\ &\quad + \frac{1-\mathbf{c}[i+1]}{2}(\Delta(i) + r_{i+1} - q_{i+1}) \\ &= \Delta(i) + \alpha_{i+1}\mathbf{c}[i+1] + \beta_{i+1},\end{aligned} \tag{12}$$

where

$$\alpha_{i+1} = \frac{p_{i+1} - r_{i+1} - s_{i+1} + q_{i+1}}{2},$$
$$\beta_{i+1} = \frac{p_{i+1} + r_{i+1} - s_{i+1} - q_{i+1}}{2}.$$

To form a compact representation for delay difference $\Delta(n-1)$ after the final stage, we calculate as follows:

$$\begin{aligned}\Delta(-1) &= 0 \\ \Delta(0) &= \Delta(-1) + \alpha_0\mathbf{c}[0] + \beta_0 = \alpha_0\mathbf{c}[0] + \beta_0 \\ \Delta(1) &= \alpha_1\mathbf{c}[1] + \alpha_0\mathbf{c}[0] + \beta_1 + \beta_0 \\ \Delta(2) &= \alpha_2\mathbf{c}[2] + \alpha_1\mathbf{c}[1] + \alpha_0\mathbf{c}[0] + \beta_2 + \beta_1 + \beta_0 \\ &\cdots \\ \Delta(n-1) &= \alpha_{n-1}\mathbf{c}[n-1] + \cdots + \alpha_0\mathbf{c}[0] + (\beta_{n-1} + \cdots + \beta_0) \\ &= \mathbf{w}[n]\mathbf{\Phi}[n] + \mathbf{w}[n-1]\mathbf{\Phi}[n-1] + \cdots + \mathbf{w}[0]\mathbf{\Phi}[0] \\ &= \mathbf{w} \cdot \mathbf{\Phi}^{\mathsf{T}},\end{aligned} \tag{13}$$

where

$$\mathbf{w}[i] = \begin{cases} \beta_{n-1} + \cdots + \beta_0 & \text{if } i = n \\ \alpha_i & \text{if } i = 0, \ldots, n-1, \end{cases}$$

$\mathbf{\Phi}[0:n-1] = \mathbf{c}[0:n-1]$, and $\mathbf{\Phi}[n] = 1$.

Hence, we conclude that the PAPUF follows a linear additive delay model, very similar to the classic APUF [2]. However, there are also important differences between the delay models of the classic APUF and the PAPUF:

1) The feature vector $\mathbf{\Phi}$ in PAPUF model is the challenge $\mathbf{c}$ itself while $\mathbf{\Phi}$ in APUF is a challenge parity vector [2].
2) All elements of $\mathbf{w}$ (except $\mathbf{w}[n]$) in PAPUF model depend only on the α values, whereas most of the elements of $\mathbf{w}$ in APUF depend on both α and β values. In addition, $\mathbf{w}[n]$ in APUF depends on the β_{n-1} value, but $\mathbf{w}[n]$ in PAPUF depends on $\beta_0, \cdots, \beta_{n-1}$. Later, this observation will be used to compare the architectural bias values of APUF and PAPUF.

Next we present a delay model for the DAPUF architecture.

C. Delay Model for DAPUF

In [21], the DAPUF was proposed to reduce the implementation bias in FPGA-based classic APUF, as shown in Fig. 4. In this scheme, two APUF instances are evaluated in parallel for a given challenge, and two 1-bit responses are generated. The upper paths of both APUF instances are connected to an arbiter circuit to compare their delays. Similarly, signal propagation delays of both the lower paths are compared using another arbiter circuit. The main assumption made for this design scheme is that FPGA CAD tool maintains similar routing for

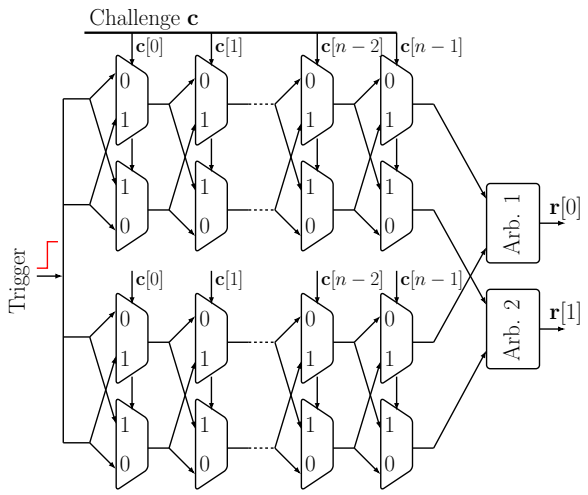


Fig. 4: A n-stage Double APUF with challenge $\mathbf{c} \in \{0,1\}^n$.

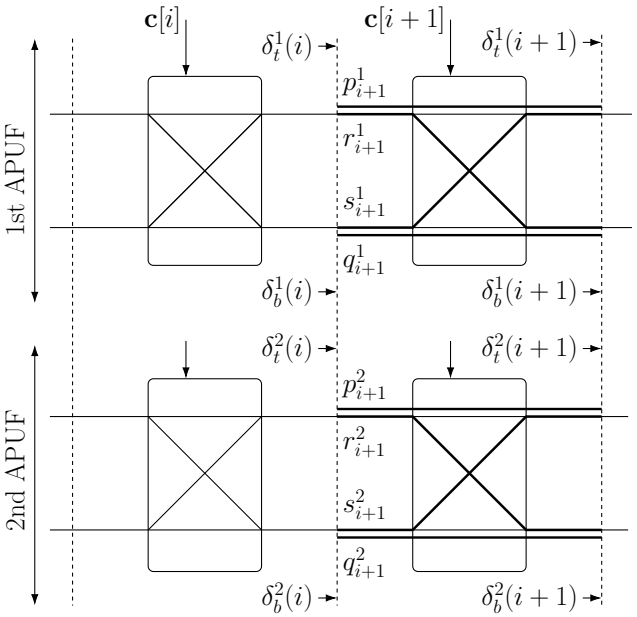


Fig. 5: The ith and $(i+1)$th switching stage of DAPUF.

upper paths of both the APUF instances with high probability, provided that they are placed in physical proximity and with similar placement constraints. Hence, it is expected that the two paths will have similar nominal propagation delays. This also holds for lower paths of both the APUF instances. If the APUF instances are instantiated twice as quasi-identical using *hard macros* (excluding the arbiters), the designer can expect elimination of the implementation bias.

Let $\delta_t^j(i)$ and $\delta_b^j(i)$ be the propagation delays of trigger signal from the starting point to the top and bottom output lines of the ith switch of jth APUF, respectively. Let p_i^j, q_i^j, r_i^j and s_i^j be the values of four delay components of the ith switching stage of jth APUF, as shown in Fig. 5. Now, we can derive the delay of the trigger signal at the output of $(i+1)$th stage with challenge $\mathbf{c} \in \{+1, -1\}$ as follows:

$$\delta_t^1(i+1) = \frac{(1-\mathbf{c}[i+1])}{2}\left[\delta_t^1(i) + p_{i+1}^1\right] \tag{14}$$

$$+\frac{(1+\mathbf{c}[i+1])}{2}\left[\delta_b^1(i)+s_{i+1}^1\right]$$
$$\delta_t^2(i+1)=\frac{(1-\mathbf{c}[i+1])}{2}\left[\delta_t^2(i)+p_{i+1}^2\right]$$
$$+\frac{(1+\mathbf{c}[i+1])}{2}\left[\delta_b^2(i)+s_{i+1}^2\right]$$
$$\delta_b^1(i+1)=\frac{(1-\mathbf{c}[i+1])}{2}\left[\delta_b^1(i)+q_{i+1}^1\right]$$
$$+\frac{(1+\mathbf{c}[i+1])}{2}\left[\delta_t^1(i)+r_{i+1}^1\right]$$
$$\delta_b^2(i+1)=\frac{(1-\mathbf{c}[i+1])}{2}\left[\delta_b^2(i)+q_{i+1}^2\right]$$
$$+\frac{(1+\mathbf{c}[i+1])}{2}\left[\delta_t^2(i)+r_{i+1}^2\right]$$

In case of DAPUF, we compute the delay difference of top paths of two APUF instances to generate the response $\mathbf{r}[0]$. Similarly, $\mathbf{r}[1]$ is generated by computing the delay difference of two bottom paths of APUFs. Below we provide only the analytical delay model to generate $\mathbf{r}[0]$, and the model for $\mathbf{r}[1]$ can be built in similar way. Let $\Delta_t(i)$ and $\Delta_t(i+1)$ be the delay differences of two top paths of the two APUF instances after the ith and $(i+1)$th stages of DAPUF, respectively. Similarly, let $\Delta_b(i)$ and $\Delta_b(i+1)$ represent the delay difference of the two bottom paths after the ith and $(i+1)$th stages, respectively. Then,

$$\begin{aligned}\Delta_t(i+1)&=\delta_t^1(i+1)-\delta_t^2(i+1) \qquad (15)\\ &=\Delta_t(i)\frac{(1-\mathbf{c}[i+1])}{2}+\Delta_b(i)\frac{(1+\mathbf{c}[i+1])}{2}\\ &\quad+\alpha_{i+1}^t\mathbf{c}[i+1]+\beta_{i+1}^t,\end{aligned}$$

where

$$\alpha_{i+1}^t=\frac{p_{i+1}^2-p_{i+1}^1+s_{i+1}^1-s_{i+1}^2}{2},$$
$$\beta_{i+1}^t=\frac{p_{i+1}^1-p_{i+1}^2+s_{i+1}^1-s_{i+1}^2}{2}$$

$$\begin{aligned}\Delta_b(i+1)&=\delta_b^1(i+1)-\delta_b^2(i+1)\\ &=\Delta_b(i)\frac{(1-\mathbf{c}[i+1])}{2}+\Delta_t(i)\frac{(1+\mathbf{c}[i+1])}{2}\\ &\quad+\alpha_{i+1}^b\mathbf{c}[i+1]+\beta_{i+1}^b,\end{aligned}$$

where

$$\alpha_{i+1}^b=\frac{q_{i+1}^2-q_{i+1}^1+r_{i+1}^1-r_{i+1}^2}{2},$$
$$\beta_{i+1}^b=\frac{q_{i+1}^1-q_{i+1}^2+r_{i+1}^1-r_{i+1}^2}{2}.$$

To find a compact representation of $\Delta_t(n-1)$, we compute different instances of $\Delta_t(i)$ as follows:

$$\Delta_t(-1)=0$$
$$\Delta_b(-1)=0$$
$$\Delta_t(0)=\alpha_0^t\mathbf{c}[0]+\beta_0^t$$
$$\Delta_b(0)=\alpha_0^b\mathbf{c}[0]+\beta_0^b$$

$$\begin{aligned}\Delta_t(1)&=\Delta_t(0)\frac{(1-\mathbf{c}[1])}{2}+\Delta_b(0)\frac{(1+\mathbf{c}[1])}{2}+\alpha_1^t\mathbf{c}[1]+\beta_1^t\\ &=\mathbf{c}[0]\frac{(\alpha_0^t+\alpha_0^b)}{2}+\mathbf{c}[1]\frac{\beta_0^b-\beta_0^t+2\alpha_1^t}{2}\\ &\quad+\mathbf{c}[0]\mathbf{c}[1]\frac{(\alpha_0^b-\alpha_0^t)}{2}+\frac{(\beta_0^t+\beta_0^b+2\beta_1^t)}{2}\end{aligned}$$

$$\begin{aligned}\Delta_b(1)&=\Delta_t(0)\frac{(1+\mathbf{c}[1])}{2}+\Delta_b(0)\frac{(1-\mathbf{c}[1])}{2}+\alpha_1^b\mathbf{c}[1]+\beta_1^b\\ &=\mathbf{c}[0]\frac{(\alpha_0^t+\alpha_0^b)}{2}+\mathbf{c}[1]\frac{\beta_0^t-\beta_0^b+2\alpha_1^b}{2}\\ &\quad+\mathbf{c}[0]\mathbf{c}[1]\frac{(\alpha_0^t-\alpha_0^b)}{2}+\frac{(\beta_0^t+\beta_0^b+2\beta_1^b)}{2}\end{aligned}$$

$$\begin{aligned}\Delta_t(2)&=\mathbf{c}[0]\frac{\alpha_0^t+\alpha_0^b}{2}+\mathbf{c}[1]\frac{\alpha_1^t+\alpha_1^b}{2}+\mathbf{c}[2]\frac{\beta_1^b-\beta_1^t+2\alpha_2^t}{2}\\ &\quad+\mathbf{c}[1]\mathbf{c}[2]\frac{\alpha_1^b-\alpha_1^t+\beta_0^t-\beta_0^b}{2}+\mathbf{c}[0]\mathbf{c}[1]\mathbf{c}[2]\frac{\alpha_0^t-\alpha_0^b}{2}\\ &\quad+\frac{\beta_0^t+\beta_0^b+\beta_1^b+\beta_1^t+2\beta_2^t}{2}\end{aligned}$$

$$\begin{aligned}\Delta_b(2)&=\mathbf{c}[1]\frac{\alpha_0^t+\alpha_0^b}{2}+\mathbf{c}[1]\frac{\alpha_1^t+\alpha_1^b}{2}+\mathbf{c}[2]\frac{\beta_1^t-\beta_1^b+2\alpha_2^b}{2}\\ &\quad+\mathbf{c}[1]\mathbf{c}[2]\frac{\alpha_1^t-\alpha_1^b+\beta_0^b-\beta_0^t}{2}+\mathbf{c}[0]\mathbf{c}[1]\mathbf{c}[2]\frac{\alpha_0^b-\alpha_0^t}{2}\\ &\quad+\frac{\beta_0^t+\beta_0^b+\beta_1^b+\beta_1^t+2\beta_2^b}{2}\end{aligned}$$

$$\begin{aligned}\Delta_t(3)&=\mathbf{c}[0]\frac{\alpha_0^t+\alpha_0^b}{2}+\mathbf{c}[1]\frac{\alpha_1^t+\alpha_1^b}{2}+\mathbf{c}[2]\frac{\alpha_2^t+\alpha_2^b}{2}\\ &\quad+\mathbf{c}[3]\frac{\beta_2^b-\beta_2^t+2\alpha_3^t}{2}+\mathbf{c}[2]\mathbf{c}[3]\frac{\alpha_2^b-\alpha_2^t+\beta_1^t-\beta_1^b}{2}\\ &\quad+\mathbf{c}[1]\mathbf{c}[2]\mathbf{c}[3]\frac{\alpha_1^t-\alpha_1^b+\beta_0^b-\beta_0^t}{2}\\ &\quad+\mathbf{c}[0]\mathbf{c}[1]\mathbf{c}[2]\mathbf{c}[3]\frac{\alpha_0^b-\alpha_0^t}{2}\\ &\quad+\frac{\beta_0^t+\beta_0^b+\beta_1^b+\beta_1^t+\beta_2^b+\beta_2^t+2\beta_3^t}{2}\end{aligned}$$

$$\begin{aligned}\Delta_b(3)&=\mathbf{c}[0]\frac{\alpha_0^t+\alpha_0^b}{2}+\mathbf{c}[1]\frac{\alpha_1^t+\alpha_1^b}{2}+\mathbf{c}[2]\frac{\alpha_2^t+\alpha_2^b}{2}\\ &\quad+\mathbf{c}[3]\frac{\beta_2^t-\beta_2^b+2\alpha_3^b}{2}+\mathbf{c}[2]\mathbf{c}[3]\frac{\alpha_2^t-\alpha_2^b+\beta_1^b-\beta_1^t}{2}\\ &\quad+\mathbf{c}[1]\mathbf{c}[2]\mathbf{c}[3]\frac{\alpha_1^b-\alpha_1^t+\beta_0^t-\beta_0^b}{2}\\ &\quad+\mathbf{c}[0]\mathbf{c}[1]\mathbf{c}[2]\mathbf{c}[3]\frac{\alpha_0^t-\alpha_0^b}{2}\\ &\quad+\frac{\beta_0^t+\beta_0^b+\beta_1^b+\beta_1^t+\beta_2^b+\beta_2^t+2\beta_3^b}{2}\end{aligned}$$

We define:

$$\mathbf{p}[i]=\begin{cases}\prod_{j=i+1}^{n}\mathbf{c}[i] & \text{if } 0\le i<n\\ 1 & \text{if } i=n.\end{cases}$$

Let us consider another two vectors $\mathbf{x}[0:n-1]$ and $\mathbf{y}[0:n-1]$, which are defined as follows:

$$\mathbf{x}_t[i]=\begin{cases}\frac{\alpha_i^t+\alpha_i^b}{2} & \text{if } 0\le i<n-1\\ \frac{\beta_b^{i-1}-\beta_t^{i-1}+2\alpha_i^t}{2} & \text{if } i=n-1\end{cases}$$

$$\mathbf{y}_t[0]=\begin{cases}\frac{\alpha_0^b-\alpha_0^t}{2} & \text{if } n \text{ is even}\\ \frac{\alpha_0^t-\alpha_0^b}{2} & \text{if } n \text{ is odd}\end{cases}$$

For $1\le i<n-1$,

$$\mathbf{y}_t[i] = \begin{cases} \frac{\alpha_i^b - \alpha_i^t + \beta_{i-1}^t - \beta_{i-1}^b}{2} & \text{if } n \text{ even and } i \text{ is odd} \\ \frac{\alpha_i^t - \alpha_i^b + \beta_{i-1}^b - \beta_{i-1}^t}{2} & \text{if } n \text{ even and } i \text{ is even} \\ \frac{\alpha_i^t - \alpha_i^b + \beta_{i-1}^b - \beta_{i-1}^t}{2} & \text{if } n \text{ odd and } i \text{ is odd} \\ \frac{\alpha_i^b - \alpha_i^t + \beta_{i-1}^t - \beta_{i-1}^b}{2} & \text{if } n \text{ odd and } i \text{ is even} \end{cases}$$

$$\mathbf{y}_t[n-1] = \frac{\beta_0^t + \beta_0^b + \cdots + \beta_{n-2}^t + \beta_{n-2}^b + 2\beta_{n-1}^t}{2}$$

So, $\Delta_t(n-1)$ can be represented as:

$$\Delta_t(n-1) = \mathbf{x}_t \cdot \mathbf{c}^\mathsf{T} + \mathbf{y}_t \cdot \mathbf{p}^\mathsf{T} = \mathbf{w} \cdot \mathbf{\Phi}^\mathsf{T}, \tag{16}$$

where $\mathbf{w}[0:2n-1] = (\mathbf{x}_t, \mathbf{y}_t)$ and $\mathbf{\Phi}[0:2n-1] = (\mathbf{c}, \mathbf{p})$. Similarly, we can compute $\Delta_b(n-1)$, the delay difference of two bottom paths of DAPUF. We can define the two responses of DAPUF by using the proposed delay model as follows:

$$\mathbf{r}[0] = \begin{cases} 1 & \text{if } \Delta_t(n-1) < 0, \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathbf{r}[1] = \begin{cases} 1 & \text{if } \Delta_b(n-1) < 0, \\ 0 & \text{otherwise.} \end{cases}$$

The above analysis shows that each individual output of a DAPUF can be estimated using a linear additive delay model. In case of DAPUF, the number of delay parameters to be learned is twice that of an APUF. Also, the delay difference expression is an inner product involving both the challenge vector and the parity vector. Thus, we can say that the delay model of a DAPUF is a combination of those for a classical APUF and a PAPUF.

Next we evaluate the architectural bias of the three different PUF variants (classic APUF, PAPUF and DAPUF), using the delay models derived above.

V. Architectural Bias and its Application As a Metric to Evaluate APUF, PAPUF and DAPUF Designs

Let the delay components p_i, q_i, r_i, s_i of the ith path-swapping switch be independent and identically distributed random variables, each of which follow normal distribution with mean $\mu = 0$ and variance σ^2, i.e., $p_i, q_i, r_i, s_i \sim \mathcal{N}(0, \sigma^2)$ [2]. Let $Z = \Delta(n-1)$ be a random variable representing the delay difference of an n stage delay PUF instance, and $Z \sim \mathcal{N}(\mu_\mathrm{Z}, \sigma_\mathrm{Z}^2)$. Based on the models built so far as in Eqs. (9), (13) and (16), we can write:

$$\begin{aligned} Z &= \left(\sum_{i=0}^{k-1} \mathbf{w}[i]\Phi[i] \right) + \mathbf{w}[k] \\ &= X + Y, \end{aligned} \tag{17}$$

where $X = \sum_{i=0}^{k-1} \mathbf{w}[i]\Phi[i]$, $Y = \mathbf{w}[k]$, and k is the size of weight vector $\mathbf{w}$ in the definition of $\Delta(n-1)$. Here, $X \sim \mathcal{N}(0, \sigma_\mathrm{X}^2)$ and $Y \sim \mathcal{N}(0, \sigma_\mathrm{Y}^2)$ are Gaussian random variables. The values of parameters σ_X^2 and σ_Y^2 depend on the architecture of APUF, PAPUF and DAPUF that will be discussed later in this section. The key difference between X and Y is that while X depends on the challenge $\mathbf{c}$ and process variations of the device where the PUF is embedded, Y only depends on the process variations of the device.

Note that for a specific APUF instance, $Y = y$ is a constant because it depends only on the delay parameters of PUF instance, and consequently $Z = X + y \sim \mathcal{N}(y, \sigma_\mathrm{X}^2)$, where $\sigma_\mathrm{Z}^2 = \sigma_\mathrm{X}^2$, assuming equiprobable challenges. According to Eqs. (5) and (6), an ideal PUF should achieve $\mu_\mathrm{Z} = 0$ instead of $\mu_\mathrm{Z} = y (\neq 0)$, and this shift in μ_Z happens due to the architectural weakness of PUF design. We term this resulting bias in the behavior of a PUF instance as *architectural bias* and it is defined as follows:

Definition 1 (Architectural Bias). *Let Z be a random variable representing the delay difference of an ideal (without any implementation-induced bias) PUF instance of a given type, and suppose it follows $\mathcal{N}(y, \sigma_\mathrm{Z}^2)$, where y is a particular value assumed by the random variable $Y \sim \mathcal{N}(0, \sigma_\mathrm{Y}^2)$ as defined in Eq. (17). Then, the value y is the challenge independent, but architecture dependent and instance-specific component in delay difference. Architectural bias of a PUF instance is defined as $b_\mathrm{archi} = |0.5 - B|$, where $B = \Phi(\frac{-y}{\sigma_\mathrm{Z}})$ is the uniformity metric as defined in Eq. (6). It is desirable that a good PUF design should have almost zero architecture induced bias (i.e. $b_\mathrm{archi} \approx 0$), for most of its instances in a given population.*

To simplify the notation, we use b to imply the architectural bias b_archi. We now consider a population of instances of a given PUF design, and evaluate its quality by estimating the percentage of "good" and "bad" instances. Since it is difficult to design a PUF with $b = 0$, we introduce an acceptable *tolerance* in the bias value and we denote it by $0 < \epsilon < 1$. This implies $0 \leq b \leq \epsilon$ or equivalently,

$$0.5 - \epsilon \leq B \leq 0.5 + \epsilon. \tag{18}$$

Based on Eq. (18), we now estimate a range for y that can ensure $0 \leq b \leq \epsilon$, as follows:

$$\begin{aligned} & 0.5 - \epsilon \leq B \leq 0.5 + \epsilon \\ & \Rightarrow \Phi^{-1}(0.5 - \epsilon) \leq \frac{-y}{\sigma_\mathrm{Z}} \leq \Phi^{-1}(0.5 + \epsilon) \\ & \Rightarrow -\Phi^{-1}(0.5 + \epsilon) \leq \frac{y}{\sigma_\mathrm{Z}} \leq -\Phi^{-1}(0.5 - \epsilon) \\ & \Rightarrow -\sigma_\mathrm{Z}\Phi^{-1}(0.5 + \epsilon) \leq y \leq -\sigma_\mathrm{Z}\Phi^{-1}(0.5 - \epsilon). \end{aligned} \tag{19}$$

Due to the symmetric nature of the Gaussian distribution, we can write $|\sigma_\mathrm{Z}\Phi^{-1}(0.5+\epsilon)| = |\sigma_\mathrm{Z}\Phi^{-1}(0.5-\epsilon)|$. Let us denote $T_\mathrm{y} = \sigma_\mathrm{Z}\Phi^{-1}(0.5 + \epsilon)$ to be the threshold for acceptable y values. Thus, we have:

$$-T_y \leq y \leq T_y. \tag{20}$$

In practice, the acceptable uniformity range of a PUF design is $40\% - 60\%$, and thus practical value of $\epsilon = 0.1$. Hence, practical value for T_y is:

$$T_\mathrm{y} = \sigma_\mathrm{Z}\Phi^{-1}(0.5 + 0.1) = \frac{\sigma_\mathrm{Z}}{4}. \tag{21}$$

We now give concrete definition of the *goodness* of a PUF instance:

Definition 2 (Good PUF Instance vs. Bad PUF Instance). *For a given instance of a PUF design, if its y value lies in*

*$[-T_y, T_y]$, then that instance is called as "**Good**" instance with respect to architectural bias; otherwise, PUF instance is "**Bad**."*

For a given PUF design, we now estimate the fraction of "Good" PUF instances. Let us recall the well-known property of a Gaussian distribution: about 99.7% of values drawn from a Gaussian distribution lie in the range $[\mu - 3\sigma, \mu + 3\sigma]$. So, the most probable range of y is $[-3\sigma_{\rm Y}, +3\sigma_{\rm Y}]$, i.e., $Pr(y \in [-3\sigma_{\rm Y}, +3\sigma_{\rm Y}]) \approx 0.997$. Let G be the fraction of good PUF instances, and we compute it based on T_y as:

$$G = \begin{cases} \Phi_{0,\sigma_{\rm Y}^2}(T_y) - \Phi_{0,\sigma_{\rm Y}^2}(-T_y) & \\ = \Phi(\frac{\sigma_{\rm Z}}{4\sigma_{\rm Y}}) - \Phi(-\frac{\sigma_{\rm Z}}{4\sigma_{\rm Y}}), & \text{if} \quad T_y < 3\sigma_{\rm Y} \\ & \\ \Phi_{0,\sigma_{\rm Y}^2}(3\sigma_{\rm Y}) - \Phi_{0,\sigma_{\rm Y}^2}(-3\sigma_{\rm Y}) & \\ = \Phi(3) - \Phi(-3) \approx 0.997, & \text{if} \quad T_y \geq 3\sigma_{\rm Y} \end{cases} \tag{22}$$

Based on the percentage of good and bad PUF instances of a PUF design, we can define the goodness of a PUF design as follows:

Definition 3 (Good PUF Design vs. Bad PUF Design). *For a given PUF design M, if the fraction of good PUF instances $G_{\rm M}$ is greater than some threshold $G_{\rm th}$, then PUF design M is called as "**Good**" design. Otherwise, design M is "**Bad**." For example, $G_{\rm th} = 0.90$, i.e., if the PUF design M can generate 90% good PUF instances, then design M is good.*

For a pair of PUF designs (M_1, M_2), if $G_1 > G_2$, then design M_1 is considered better than M_2.

Next we apply this approach to evaluate APUF, PAPUF and DAPUF designs with their architectural biases.

A. Architectural Bias of APUF

Let $Z_{\rm A} = \Delta(n-1)$ be a random variable representing the delay difference of APUF instance and it follows a normal distribution $\mathcal{N}(\mu_{\rm A}, \sigma_{\rm A}^2)$. Based on the modeling of APUF as in Eq. (9), we can write:

$$\begin{aligned} Z_{\rm A} &= \left(\sum_{i=0}^{n-1} \mathbf{w}[i]\Phi[i]\right) + \mathbf{w}[n] \\ &= X_{\rm A} + Y_{\rm A}, \end{aligned}$$

where $X_{\rm A} = \sum_{i=0}^{n-1} \mathbf{w}[i]\Phi[i]$ and $Y_{\rm A} = \mathbf{w}[n]$. From Eq. (11), we have:

1) $\mathbf{w}[0], \mathbf{w}[n] \sim \mathcal{N}(0, 2\sigma^2)$ and $\mathbf{w}[i] \sim \mathcal{N}(0, 4\sigma^2)$, for $i = 1, \ldots, n-1$,
2) $X_{\rm A} \sim \mathcal{N}(0, \sigma_{\rm A}^2)$ where $\sigma_{\rm A}^2 = (4n-2)\sigma^2 \approx 4n\sigma^2$ for moderately large values of n, and
3) $Y_{\rm A} \sim \mathcal{N}(0, \sigma_{y_{\rm A}}^2)$ where $\sigma_{y_{\rm A}}^2 = 2\sigma^2$.

Thus, for a specific APUF instance, $Y_{\rm A} = y_{\rm A}$ is a constant and $Z_{\rm A} = X_{\rm A} + y_{\rm A} \sim \mathcal{N}(y_{\rm A}, \sigma_{\rm A}^2)$.

Based on Eq. (19), we can compute the threshold for acceptable $y_{\rm A}$ values as:

$$T_{y_{\rm A}} = \sigma_{\rm A}/4 = \sqrt{n}\sigma/2.$$

In case of the Gaussian random variable $Y_{\rm A}$, nearly 99.7% of $y_{\rm A}$ values lies in $[-3\sigma_{y_{\rm A}}, +3\sigma_{y_{\rm A}}]$, where $3\sigma_{y_{\rm A}} = 3\sqrt{2}\sigma$. Since $T_{y_{\rm A}} > 3\sigma_{y_{\rm A}}$ for $n > 72$, the fraction of good APUF instances, as defined by Eq. (22), is:

$$G_{\rm A} \approx 0.997.$$

Later, we would find from the simulation results in Table II and Fig. 8, that the distribution of the uniformity $B_{\rm A}$ of APUF has a mean 0.5, and the variance is significantly small due to the above facts. Hence, we conclude that the APUF is relatively free from architectural bias.

B. Architectural Bias of PAPUF

Let $Z_{\rm P} = \Delta(n-1)$ be a Gaussian random variable representing the delay difference of a PAPUF instance. From Eq. (13), we can write:

$$\begin{aligned} Z_{\rm P} &= \left(\sum_{i=0}^{n-1} \mathbf{w}[i]\Phi[i]\right) + \mathbf{w}[n] \\ &= X_{\rm P} + Y_{\rm P}. \end{aligned}$$

Like APUF, $X_{\rm P}$ is challenge dependent, and $Y_{\rm P}$ is challenge independent. We also have following information from Eq. (13):

1) $\mathbf{w}[i] \sim \mathcal{N}(0, 2\sigma^2)$ for $i = 0, \ldots, n-1$, and $\mathbf{w}[n] \sim \mathcal{N}(0, 2n\sigma^2)$,
2) $X_{\rm P} \sim \mathcal{N}(0, \sigma_{\rm P}^2)$ where $\sigma_{\rm P}^2 = 2n\sigma^2$, and
3) $Y_{\rm P} \sim \mathcal{N}(0, \sigma_{y_{\rm P}}^2)$ where $\sigma_{y_{\rm P}}^2 = 2n\sigma^2$.

Thus, for a specific PAPUF instance, $Y_{\rm P} = y_{\rm P}$ is a fixed value and $Z_{\rm P} = X_{\rm P} + y_{\rm P} \sim \mathcal{N}(y_{\rm P}, \sigma_{\rm P}^2)$.

Based on Eq. (19), we can compute the threshold for acceptable $y_{\rm P}$ values as:

$$T_{y_{\rm P}} = \sigma_{\rm P}/4 = \sqrt{2n}\sigma/4.$$

In case of Gaussian random variable $Y_{\rm P}$, nearly 99.7% of $y_{\rm P}$ values lies in $[-3\sigma_{y_{\rm P}}, +3\sigma_{y_{\rm P}}]$, where $3\sigma_{y_{\rm P}} = 3\sqrt{2n}\sigma$. In this case, we have $T_{y_{\rm P}} < 3\sigma_{y_{\rm P}}$. Thus, the fraction of good PAPUF instances, as defined in Eq. (22), is:

$$\begin{aligned} G_{\rm P} &= \Phi(\frac{\sigma_{\rm P}}{4\sigma_{y_{\rm P}}}) - \Phi(-\frac{\sigma_{\rm P}}{4\sigma_{y_{\rm P}}}) \\ &= \Phi(\frac{\sqrt{2n}\sigma}{4\sqrt{2n}\sigma}) - \Phi(-\frac{\sqrt{2n}\sigma}{4\sqrt{2n}\sigma}) \\ &= \Phi(1/4) - \Phi(-1/4) \approx 0.20. \end{aligned} \tag{23}$$

Thus, it is expected that the distribution of uniformity for the PAPUF, $B_{\rm P}$, should have mean close to 0.5 and significantly large variance, as also supported by our simulation results reported later in Table II and Fig. 8. Thus, we conclude that the uniformity property of the PAPUF is relatively poor. To the best of our knowledge, until now no theoretical explanation has been provided for this fact, although it was reported based on experimental observations in [20], [24].

C. Architectural Bias of DAPUF

Let $Z_{\rm D} = \Delta(n-1)$ be a Gaussian random variable representing the delay differences of a DAPUF instance. From Eq. (16), we can write

$$\begin{aligned} Z_{\rm D} &= \left(\sum_{i=0}^{2n-2} \mathbf{w}[i]\Phi[i]\right) + \mathbf{w}[n] \\ &= X_{\rm D} + Y_{\rm D}, \end{aligned}$$

From the definition of $\mathbf{w}[i]$ in Eq. (16), we have:

1) $X_{\rm D} \sim \mathcal{N}(0, \sigma_{\rm D}^2)$ where $\sigma_{\rm D}^2 = 6n\sigma^2$, and
2) $Y_{\rm D} \sim \mathcal{N}(0, \sigma_{y_{\rm D}}^2)$ where $\sigma_{y_{\rm D}}^2 = 2n\sigma^2$.

Thus, for a specific DAPUF instance, $Y_{\rm D} = y_{\rm D}$ and $Z_{\rm D} = X_{\rm D} + y_{\rm D} \sim \mathcal{N}(y_{\rm D}, \sigma_{\rm D}^2)$.

Based on Eq. (19), we can compute the threshold of acceptable $y_{\rm D}$ values as:

$$T_{y_{\rm D}} = \sigma_{\rm D}/4 = \sqrt{6n}\sigma/4.$$

In case of Gaussian random variable $Y_{\rm D}$, nearly 99.7% of $y_{\rm D}$ values lies in $[-3\sigma_{y_{\rm D}}, +3\sigma_{y_{\rm D}}]$, where $3\sigma_{y_{\rm D}} = 3\sqrt{2n}\sigma$. In this case, we have $T_{y_{\rm D}} < 3\sigma_{y_{\rm D}}$. Thus, the fraction of good DPUF instances, as defined in Eq. (22), is:

$$\begin{aligned} G_{\rm D} &= \Phi\left(\frac{\sigma_{\rm D}}{4\sigma_{y_{\rm D}}}\right) - \Phi\left(-\frac{\sigma_{\rm D}}{4\sigma_{y_{\rm D}}}\right) \qquad (24) \\ &= \Phi\left(\frac{\sqrt{6n}\sigma}{4\sqrt{2n}\sigma}\right) - \Phi\left(-\frac{\sqrt{6n}\sigma}{4\sqrt{2n}\sigma}\right) \\ &= \Phi(\sqrt{3}/4) - \Phi(-\sqrt{3}/4) \approx 0.335. \end{aligned}$$

So, the uniformity property of the DAPUF is poor in general due to the architectural weakness, though some instances might show good uniformity individually. Moreover, in case of DAPUF design, there are 13.5% more "good" instances compared to PAPUF.

Now, we provide a methodology to compare a given pair of randomly selected PUF instances from two different PUF designs using their architectural bias.

VI. Instance-Level Comparison of PUF Designs with Architectural Bias

In Section V, we have shown how to measure the quality of a PUF design based on the percentage of good PUF instances of its. In this section, we introduce a new problem as defined below.

Problem. *Let I_1 and I_2 be two randomly selected instances of two different PUF designs P_1 and P_2, respectively. Let $Z_1 \sim \mathcal{N}(y_1, \sigma_1^2)$ and $Z_2 \sim \mathcal{N}(y_2, \sigma_2^2)$ be two random variables representing the delay differences of PUF instances I_1 and I_2, respectively. Based on this information, is it possible to identify the "better" PUF instance? Can we extend this instance-level comparison, to design-level comparison of PUF, using the information of distributions from which y_1 and y_2 are drawn?*

The scheme discussed in Section V cannot be used to perform the instance-level comparison of two different PUF designs. In this section, we develop a scheme to deal with this problem. Let b_1 and b_2 be the architectural bias of PUF instances I_1 and I_2. Let us assume that instance I_1 is "better" than I_2, in the sense of the goodness described by us. Then, we derive a relationship among parameters $y_1, \sigma_1^2, y_2, \sigma_2^2$ such that $b_2 > b_1$.

Let $\mathcal{R}$ be the relationship among $y_1, \sigma_1^2, y_2, \sigma_2^2$ such that $b_2 > b_1$. Based on $\mathcal{R}$, we can compare two PUF designs using following steps:

1) Firstly, compute the probability of the event that relation $\mathcal{R}$ holds between pair of randomly selected instances from PUF design P_1 and P_2. In other words, compute $Pr(\mathcal{R})$ with respect to P_1 and P_2.
2) Then, we can make two following decisions:
   a) PUF design P_1 is better than P_2 if $Pr(\mathcal{R}) > \frac{1}{2}$, and the reverse holds if $Pr(\mathcal{R}) < \frac{1}{2}$.
   b) PUF designs P_1 and P_2 are equivalent if $Pr(\mathcal{R}) = \frac{1}{2}$.

Next we employ this methodology to perform pairwise instance-level and design-level comparison of APUF, PAPUF and DAPUF designs.

A. APUF vs. DAPUF

We use same notations for APUF and DAPUF that we already used in Section V. Let $Y_{\rm D} = y_{\rm D}$, and $Y_{\rm A} = y_{\rm A}$. In this case, $Y_{\rm A} \sim \mathcal{N}(0, 2\sigma^2)$ and $Y_{\rm D} \sim \mathcal{N}(0, 2n\sigma^2)$; $Z_{\rm A} \sim \mathcal{N}(y_{\rm A}, \sigma_{\rm A}^2)$ and $Z_{\rm D} \sim \mathcal{N}(y_{\rm D}, \sigma_{\rm D}^2)$; $\sigma_{\rm A}^2 = 4n\sigma^2$, $\sigma_{\rm D}^2 = 6n\sigma^2$ and $\sigma_{\rm D}^2 > \sigma_{\rm A}^2$. We compute the bias values of APUF and DAPUF as:

$$b_{\rm D} = \left|\frac{1}{2} - \Phi\left(\frac{-y_{\rm D}}{\sigma_{\rm D}}\right)\right| \quad \text{and} \quad b_{\rm A} = \left|\frac{1}{2} - \Phi\left(\frac{-y_{\rm A}}{\sigma_{\rm A}}\right)\right|. \qquad (25)$$

Now, we try to find the relation between $y_{\rm A}$ and $y_{\rm D}$ such that $b_{\rm D} > b_{\rm A}$, i.e., APUF is better than DAPUF. To find the relationship between $y_{\rm A}$ and $y_{\rm D}$, we consider following four cases with respect to Eq. (25).

CASE-1. In this case, we consider:

$$\Phi\left(\frac{-y_{\rm D}}{\sigma_{\rm D}}\right) > \frac{1}{2} \Rightarrow \Phi\left(\frac{-y_{\rm D}}{\sigma_{\rm D}}\right) > \Phi(0) \Rightarrow -y_{\rm D} > 0 \Rightarrow y_{\rm D} < 0,$$

and

$$\Phi\left(\frac{-y_{\rm A}}{\sigma_{\rm A}}\right) > \frac{1}{2} \Rightarrow \Phi\left(\frac{-y_{\rm A}}{\sigma_{\rm A}}\right) > \Phi(0) \Rightarrow -y_{\rm A} > 0 \Rightarrow y_{\rm A} < 0.$$

Then, we have:

$$\begin{aligned} b_{\rm D} - b_{\rm A} &= -\frac{1}{2} + \Phi\left(\frac{-y_{\rm D}}{\sigma_{\rm D}}\right) + \frac{1}{2} - \Phi\left(\frac{-y_{\rm A}}{\sigma_{\rm A}}\right) \\ &= \Phi\left(\frac{-y_{\rm D}}{\sigma_{\rm D}}\right) - \Phi\left(\frac{-y_{\rm A}}{\sigma_{\rm A}}\right). \end{aligned}$$

If $b_{\rm D} - b_{\rm A} > 0$, then

$$\frac{-y_{\rm D}}{\sigma_{\rm D}} > \frac{-y_{\rm A}}{\sigma_{\rm A}} \Rightarrow \frac{y_{\rm D}}{\sigma_{\rm D}} < \frac{y_{\rm A}}{\sigma_{\rm A}},$$

where $y_{\rm A}, y_{\rm D} < 0$.

CASE-2. In this case, we consider:

$$\Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) < \frac{1}{2} \Rightarrow \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) < \Phi(0) \Rightarrow -y_{\rm D} < 0 \Rightarrow y_{\rm D} > 0,$$

and

$$\Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) < \frac{1}{2} \Rightarrow \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) < \Phi(0) \Rightarrow -y_{\rm A} < 0 \Rightarrow y_{\rm A} > 0,$$

Then, we have:

$$\begin{aligned} b_{\rm D} - b_{\rm A} &= \frac{1}{2} - \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) - \frac{1}{2} + \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) \\ &= \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) - \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}). \end{aligned}$$

If $b_{\rm D} - b_{\rm A} > 0$, then:

$$\frac{-y_{\rm A}}{\sigma_{\rm A}} > \frac{-y_{\rm D}}{\sigma_{\rm D}} \Rightarrow \frac{y_{\rm A}}{\sigma_{\rm A}} < \frac{y_{\rm D}}{\sigma_{\rm D}},$$

where $y_{\rm A}, y_{\rm D} > 0$.

CASE-3. In this case, we consider:

$$\Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) < \frac{1}{2} \Rightarrow \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) < \Phi(0) \Rightarrow -y_{\rm D} < 0 \Rightarrow y_{\rm D} > 0,$$

and

$$\Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) > \frac{1}{2} \Rightarrow \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) > \Phi(0) \Rightarrow -y_{\rm A} > 0 \Rightarrow y_{\rm A} < 0,$$

Then, we have:

$$\begin{aligned} b_{\rm D} - b_{\rm A} &= \frac{1}{2} - \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) + \frac{1}{2} - \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) \\ &= (1 - \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}})) - \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) \\ &= \Phi(\frac{y_{\rm D}}{\sigma_{\rm D}}) - \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}). \end{aligned}$$

If $b_{\rm D} - b_{\rm A} > 0$, then:

$$\frac{y_{\rm D}}{\sigma_{\rm D}} > \frac{-y_{\rm A}}{\sigma_{\rm A}} \Rightarrow \frac{-y_{\rm D}}{\sigma_{\rm D}} < \frac{y_{\rm A}}{\sigma_{\rm A}},$$

where $y_{\rm A} < 0, y_{\rm D} > 0$.

CASE-4. In this case, we consider:

$$\Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) > \frac{1}{2} \Rightarrow \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) > \Phi(0) \Rightarrow -y_{\rm D} > 0 \Rightarrow y_{\rm D} < 0,$$

and

$$\Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) < \frac{1}{2} \Rightarrow \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) < \Phi(0) \Rightarrow -y_{\rm A} < 0 \Rightarrow y_{\rm A} > 0,$$

Then, we have:

$$\begin{aligned} b_{\rm D} - b_{\rm A} &= -\frac{1}{2} + \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) - \frac{1}{2} + \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}}) \\ &= \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) - (1 - \Phi(\frac{-y_{\rm A}}{\sigma_{\rm A}})) \\ &= \Phi(\frac{-y_{\rm D}}{\sigma_{\rm D}}) - \Phi(\frac{y_{\rm A}}{\sigma_{\rm A}}). \end{aligned}$$

If $b_{\rm D} - b_{\rm A} > 0$, then:

$$\frac{-y_{\rm D}}{\sigma_{\rm D}} > \frac{y_{\rm A}}{\sigma_{\rm A}} \Rightarrow \frac{y_{\rm D}}{\sigma_{\rm D}} < \frac{-y_{\rm A}}{\sigma_{\rm A}},$$

where $y_{\rm A} > 0, y_{\rm D} < 0$.

Hence, considering the four cases, overall if,

$$\begin{aligned} &|y_{\rm D}/\sigma_{\rm D}| > |y_{\rm A}/\sigma_{\rm A}| \quad \text{or} \\ &|y_{\rm D}| > \frac{\sigma_D}{\sigma_A} \times |y_{\rm A}| = \frac{\sqrt{6n}\sigma}{\sqrt{4n}\sigma} \times |y_{\rm A}| = \sqrt{3/2}|y_{\rm A}|, \end{aligned}$$

then bias in APUF is less compared to that of DAPUF. It may be noted that the relation $|y_{\rm D}| > \sqrt{3/2}|y_{\rm A}|$ is referred to as $\mathcal{R}$ in this section earlier.

Now we proceed for design-level comparison by computing $Pr(|y_{\rm D}| > \sqrt{3/2}|y_{\rm A}|)$. For the sake of explanation, we denote $X = Y_{\rm D} \sim \mathcal{N}(0, 2n\sigma^2)$ and $Y = \sqrt{3/2}Y_{\rm A} \sim \mathcal{N}(0, 3\sigma^2)$, and we compute $Pr(|X| > |Y|)$. We first consider the case $Pr(X > Y | Y > 0)$, whereby we have:

$$\begin{aligned} Pr(X > Y | Y > 0) &= \int_0^{+\infty} \phi_{0,3\sigma^2}(y) \int_y^{+\infty} \phi_{0,2n\sigma^2}(x)\, {\rm d}x\, {\rm d}y \\ &= \int_0^{+\infty} \frac{1}{\sqrt{3}\sigma} \phi(\frac{y}{\sqrt{3}\sigma}) \Phi(\frac{-y}{\sqrt{2n}\sigma})\, {\rm d}y. \end{aligned}$$

Since we need to compute the above relation with respect to absolute values of the random variables, remaining cases can be computed in similar fashion. Hence, we have:

$$Pr(|X| > |Y|) = 4 \int_0^{+\infty} \frac{1}{\sqrt{3}\sigma} \phi\left(\frac{y}{\sqrt{3}\sigma}\right) \Phi\left(\frac{-y}{\sqrt{2n}\sigma}\right) {\rm d}y. \tag{26}$$

Since nearly 99.7% of values drawn from the Gaussian distribution of a random variable Y are within the $\pm 3\sigma$ range around the mean, we have following observations:

1) $\phi(\frac{y}{\sqrt{3}\sigma}) \approx 0$ when $y > 3\sqrt{3}\sigma$.
2) $\Phi(\frac{-y}{\sqrt{2n}\sigma}) \approx \Phi(0) = 1/2$ when $0 \leq y \leq 3\sqrt{3}\sigma$ and $\sqrt{2n}\sigma \gg 3\sqrt{3}\sigma$ (or $n \gg 27/2 \approx 14$).

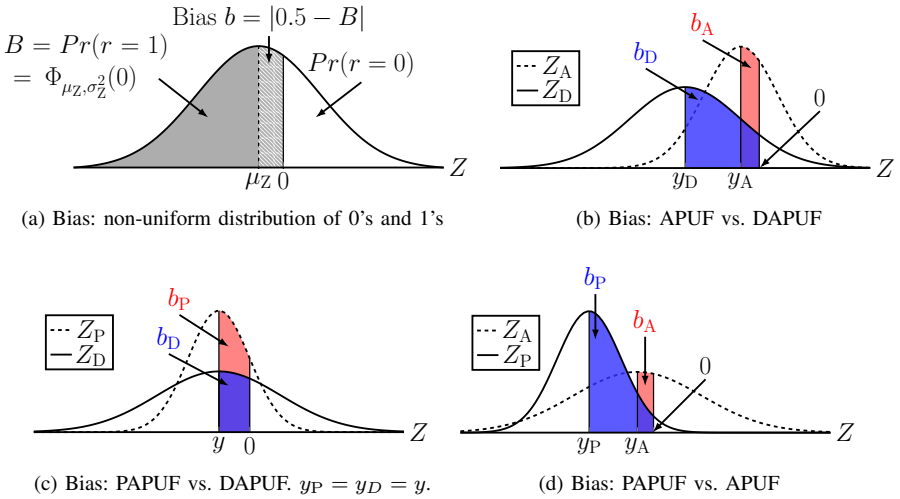
Thus, we can rewrite Eq. (26) as given below by splitting range $[0, +\infty]$ of y into $[0, +3\sqrt{3}\sigma]$ and $[+3\sqrt{3}\sigma, +\infty]$:

$$\begin{aligned} Pr(|X| > |Y|) &= 4 \int_0^{3\sqrt{3}\sigma} \frac{1}{\sqrt{3}\sigma} \phi\left(\frac{y}{\sqrt{3}\sigma}\right) \Phi\left(\frac{-y}{\sqrt{2n}\sigma}\right) {\rm d}y \\ &\quad + 4 \int_{3\sqrt{3}\sigma}^{+\infty} \frac{1}{\sqrt{3}\sigma} \phi\left(\frac{y}{\sqrt{3}\sigma}\right) \Phi\left(\frac{-y}{\sqrt{2n}\sigma}\right) {\rm d}y \\ &\approx 4 \int_0^{3\sqrt{3}\sigma} \frac{1}{\sqrt{3}\sigma} \phi\left(\frac{y}{\sqrt{3}\sigma}\right) \Phi\left(\frac{-y}{\sqrt{2n}\sigma}\right) {\rm d}y \\ &\approx 4 \times \Phi(0) \times \int_0^{3\sqrt{3}\sigma} \frac{1}{\sqrt{3}\sigma} \phi\left(\frac{y}{\sqrt{3}\sigma}\right) {\rm d}y \\ &\approx 4 \times \Phi(0) \times \int_0^{+\infty} \frac{1}{\sqrt{3}\sigma} \phi\left(\frac{y}{\sqrt{3}\sigma}\right) {\rm d}y \\ &= 4 \times \frac{1}{2} \times \frac{1}{2} = 1. \end{aligned}$$

Since $Pr(|y_{\rm D}| > \sqrt{3/2}|y_{\rm A}|) \approx 1$, we can conclude that the APUF design is better than DAPUF. Without loss of generality, we provide an example in Fig. 6b for $y_{\rm D} < y_{\rm A} < 0$.

B. PAPUF vs. DAPUF

In this case, $Y_{\rm P}, Y_{\rm D} \sim \mathcal{N}(0, 2n\sigma^2)$, $Z_{\rm P} \sim \mathcal{N}(y_{\rm P}, \sigma_{\rm P}^2)$, $Z_{\rm D} \sim \mathcal{N}(y_{\rm D}, \sigma_{\rm D}^2)$, where $\sigma_{\rm D}^2 = 6n\sigma^2$ and $\sigma_{\rm P}^2 = 2n\sigma^2$. Proceeding along similar route as used for the "DAPUF vs.



(a) Bias: non-uniform distribution of 0's and 1's

(b) Bias: APUF vs. DAPUF

(c) Bias: PAPUF vs. DAPUF. $y_P = y_D = y$.

(d) Bias: PAPUF vs. APUF

Fig. 6: Example of instance-level pairwise comparison of APUF, PAPUF and DAPUF designs with respect to delay difference distribution Z. (a) Depicting bias in the distribution of 0's and 1's due to shift in the mean of delay difference distribution from its ideal value $\mu_Z = 0$. (b) Showing the bias values of two specific instances of APUF and DAPUF with y_A and y_D as the mean values of their delay difference distributions, respectively. (c) Comparison of bias values of two specific instances of PAPUF and DAPUF with similar mean values for their delay difference distributions, i.e., $y_P = y_D = y$. (d) Comparison of bias values of two specific instances of PAPUF and APUF.

APUF" analysis, we can prove that if $|y_P| > \sqrt{3}|y_D|$, then $b_P > b_D$. The probability:

$$Pr\left(|Y_P| > \sqrt{3}|Y_D|\right) = 4 \times \int_0^{+\infty} \frac{1}{\sqrt{2n\sigma}} \phi\left(\frac{x}{\sqrt{2n\sigma}}\right) \Phi\left(\frac{-x}{\sqrt{6n\sigma}}\right) \mathrm{d}x.$$

Note that:

1) $\forall x > 0, \frac{-x}{\sqrt{6n\sigma}} > \frac{-x}{\sqrt{2n\sigma}}$.
2) If X and Y follow the same normal distribution $\mathcal{N}(0, \sigma)$, then,

$$Pr(|X| > |Y|) = 4 \times \int_0^{+\infty} \frac{1}{\sigma} \phi\left(\frac{x}{\sigma}\right) \Phi\left(\frac{-x}{\sigma}\right) \mathrm{d}x = 4 \times \frac{1}{8} = \frac{1}{2}.$$

Thus,

$$Pr\left(|Y_P| > \sqrt{3}|Y_D|\right) > 4 \times \int_0^{+\infty} \frac{1}{\sqrt{2n\sigma}} \phi\left(\frac{x}{\sqrt{2n\sigma}}\right) \Phi\left(\frac{-x}{\sqrt{2n\sigma}}\right) \mathrm{d}x = \frac{1}{2}.$$

Since, for large value of n, distance between the probability density functions of two normal distributions, say $\mathcal{N}(0, 6n\sigma^2)$ and $\mathcal{N}(0, 4n\sigma^2)$, is significantly small, we can have following approximations: $\mathcal{N}(0, 6n\sigma^2) \approx \mathcal{N}(0, 4n\sigma^2)$ and $\mathcal{N}(0, 2n\sigma^2) \approx \mathcal{N}(0, 4n\sigma^2)$. Thus, we have:

$$Pr\left(|Y_P| > \sqrt{3}|Y_D|\right) \approx 4 \times \int_0^{+\infty} \frac{1}{\sqrt{4n\sigma}} \phi\left(\frac{x}{\sqrt{4n\sigma}}\right) \Phi\left(\frac{-x}{\sqrt{4n\sigma}}\right) \mathrm{d}x = \frac{1}{2}.$$

Thus, we can conclude that DAPUF design is better then PAPUF, but for the case of large n (challenge size) they seem to be equivalent. Without loss of generality, a simple example is provided in Fig. 6c where $y_P = y_D = y < 0$.

C. PAPUF vs. APUF

In this case, $Y_A \sim \mathcal{N}(0, 2\sigma^2)$, $Y_P \sim \mathcal{N}(0, 2n\sigma^2)$, $Z_A \sim \mathcal{N}(y_A, \sigma_A^2)$, $Z_P \sim \mathcal{N}(y_P, \sigma_P^2)$, $\sigma_P^2 = 2n\sigma^2$ and $\sigma_A^2 = 4n\sigma^2$. With a similar analysis for "DAPUF vs. APUF", we can demonstrate $b_P > b_A$ because $Pr(|y_P| > |y_A|) \approx 1$ when $n \gg 3$. Thus, the APUF is better than PAPUF. Without loss of generality, we provide an example in Fig. 6d for $y_P < y_A < 0$.

VII. Impact of Architectural Bias on PUF Quality Metrics

According to Definition 1, uniformity is directly related to the architectural bias, i.e., a PUF instance with less architectural bias implies that PUF instance has good uniformity property. So, in this section, we discuss only the effects of architectural bias on uniqueness and reliability properties.

A. Uniqueness Metric

The *uniqueness* metric is used as an estimate of the average dissimilarity in the challenge-response behavior of the PUF instances. It can be defined as an average pairwise Hamming distance between the responses of a population of PUF instances of the same type, for the same set of challenges. Let P_1 and P_2 be two PUF instances with uniformity B_1 and B_2, respectively. Let r_1 and r_2 be the responses of P_1 and P_2, respectively. Assuming that the two PUF instances are independent, the uniqueness is computed as follows:

$$\begin{aligned} U &= Pr(r_1 \neq r_2) \\ &= Pr(r_1 = 0 \wedge r_2 = 1) + Pr(r_1 = 1 \wedge r_2 = 0) \\ &= (1 - B_1)B_2 + B_1(1 - B_2) = B_1 + B_2 - 2B_1B_2. \end{aligned} \tag{27}$$

We can extend this definition for multiple instances of a PUF design. As discussed in Section VI, the uniformity of APUF

is good and the uniformities of PAPUF and DAPUF are poor. Thus, according to Eq. (27), the uniqueness of an APUF is superior than both ideal DAPUF and PAPUF, and uniqueness of DAPUF is comparatively better than PAPUF. But in practice, bias-free design of APUF is almost infeasible on FPGAs; as a result FPGA based APUF shows poor uniqueness. Note that FPGA based APUF instances usually exhibit good uniformity, but poor uniqueness. One reason could be that PUF instances implemented on the FPGAs are not independent due to implementation bias. To summarize, from a theoretical (ideal) point-of-view, none of the the alternative delay PUF design variants is superior to the APUF with respect to uniqueness, but in practice, on FPGA platforms DAPUF shows better uniqueness compared to APUF due to more implementation bias in APUF.

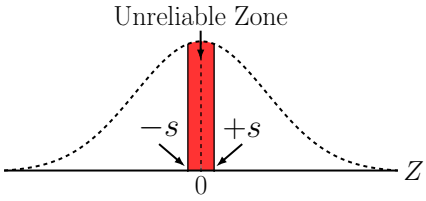
B. Reliability Metric

The reliability of a PUF is the ability to generate the same response to a given challenge repeatedly over different time instants and operating conditions, e.g. variations in ambient temperature and supply voltage. If the delay difference is significantly small, then the polarity of delay difference for a given challenge $\mathbf{c}$ might change due to noise with high probability. In addition, the behavior of the arbiter circuit used also plays an important role in determining the reliability. Typically, the arbiter is implemented using a D flip-flop (DFF). If the delay difference is less than the setup time of the DFF, then DFF enters into a metastable mode, and response to the challenge becomes noisy. Interested readers are referred to [25] for more details regarding the reliability of APUF.

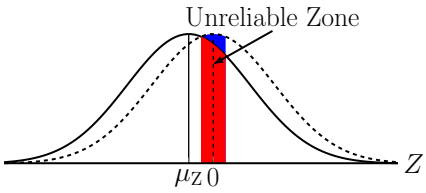
In practice, a delay difference range is estimated as shown in Fig. 7a, to determine the condition under which the PUF instance might behave unreliably. If the delay difference $\Delta_{\mathbf{c}}$ due to a given challenge $\mathbf{c}$ lies in the interval $\Delta_{\mathbf{c}} \in [-s, +s]$, then response to the challenge $\mathbf{c}$ is treated as unreliable. Value of s is decided based on the setup time of the DFF arbiter and the characterization of PUF at different operating conditions to observe the change in polarity of delay difference. In [10], [25], authors discussed about the estimation of parameter s based on the weight vector of linear delay model obtained using machine learning modeling of APUF.

Figure 7b depicts an interesting fact that if the mean μ_{Z} of delay difference distribution Z of a PUF instance is shifted from its ideal value $\mu_{\mathrm{Z}} = 0$, then the unreliable zone under the curve of Z is reduced. So, if $\mu_{\mathrm{Z}} \approx 0$ for a PUF instance, then its reliability property is poor compared to the PUF instance having $|\mu_{\mathrm{Z}}| > 0$. Based on this fact, now we can compare the reliability property of APUF ($Z_{\mathrm{A}} \sim \mathcal{N}(\mu_{\mathrm{A}}, \sigma_{\mathrm{A}}^2)$), PAPUF ($Z_{\mathrm{P}} \sim \mathcal{N}(\mu_{\mathrm{P}}, \sigma_{\mathrm{P}}^2)$) and DAPUF ($Z_{\mathrm{D}} \sim \mathcal{N}(\mu_{\mathrm{D}}, \sigma_{\mathrm{D}}^2)$).

Intuitively, if a PUF instance has *more architectural bias*, then it is *more reliable*. i.e., μ_{Z} drifts from its ideal value 0. Since, in case of APUF design, $Pr(\mu_{\mathrm{A}} \approx 0)$ and $Pr(|\mu_{\mathrm{A}}| < |\mu_{\mathrm{P}}|, |\mu_{\mathrm{D}}|)$ are significantly large, the reliability of APUF is significantly poor compared to PAPUF and DAPUF. In Section VI-B, we have shown that bias in a PAPUF instance is larger than that of a DAPUF instance when the mean values of their delay distributions follow $|\mu_{\mathrm{P}}| > \sqrt{3}|\mu_{\mathrm{D}}|$. In this scenario, PAPUF is more reliable than DAPUF. Again



(a) Unreliable zone in the delay difference distribution of a delay PUF instance without architectural bias.



(b) Reduced unreliable zone under the curve of Z due to shift in μ_{Z} from its ideal value 0. Area filled with blue color is the amount of reduction happened in the unreliable zone.

Fig. 7: Demonstration of reliability of a delay PUF instance. (a) If delay difference $\Delta_{\mathbf{c}}$ of challenge $\mathbf{c}$ lies in $[-s, +s]$, then response to challenge $\mathbf{c}$ is unreliable. This boundary is decided based on the arbiter setup time and characterization of PUF at different operating conditions. (b) Showing the shift in the mean of delay difference distribution Z due to architectural issue, and it results in the reduction of unreliable area under the curve of Z (solid-dark line).

TABLE I: Summary of quality metrics of APUF variants for arbitrary challenge size (n) without any implementation issues

Metric	Relation
Arch. Bias	$b_{\mathrm{A}} < b_{\mathrm{D}} \leq b_{\mathrm{P}}$
Uniformity	$B_{\mathrm{A}} > B_{\mathrm{D}} \geq B_{\mathrm{P}}$
Uniqueness	$U_{\mathrm{A}} > U_{\mathrm{D}} \geq U_{\mathrm{P}}$
Reliability	$S_{\mathrm{A}} < S_{\mathrm{D}} \leq S_{\mathrm{P}}$

according to Section VI-B, for large instances of PAPUF and DAPUF, they would have similar reliability for most of the PUF instances.

The relative orders of quality metrics of APUF, PAPUF and DAPUF are summarized in Table I.

VIII. Simulation and Experimental Results

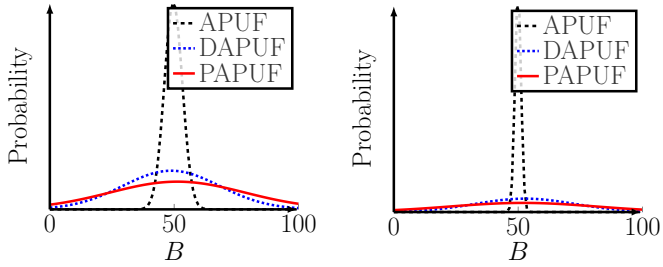
To validate the notion of architectural bias and its effects on the quality metrics of APUF, PAPUF and DAPUF, in this section we present simulation and experimental results (on FPGA platform) for these three PUF variants.

We simulated the PUFs using *Matlab* (for challenge sizes of 64-bit, 128-bit, 256-bit and 512-bit), assuming that each propagation delay of a delay component follows normal distribution with $\mu = 0$ and $\sigma = 0.05$. In case of simulation, we directly compare the delays of the top and bottom paths using an ideal comparator, so there is no impact of the non-ideal arbiter component on the PUF behavior. This helps to elucidate the impact of architectural bias, which is the focus of this paper.

TABLE II: Uniformity and uniqueness metrics of APUF design variants

	Size	PUF	Quality Metrics (μ, σ) [%]	
			Uniformity	Uniqueness
SIMULATED	64	APUF	(49.71,3.60)	(50.00,4.01)
		DAPUF	(46.89,20.07)	(50.01,8.36)
			(47.78,20.22)	(50.08,8.38)
		PAPUF	(49.07,31.46)	(50.40,19.39)
	128	APUF	(49.84,2.68)	(49.94,2.87)
		DAPUF	(48.34,19.55)	(50.20,7.65)
			(48.52,19.48)	(50.23,7.61)
		PAPUF	(55.41,24.85)	(49.60,12.64)
	256	APUF	(49.57,1.71)	(49.98,2.03)
		DAPUF	(49.74,16.93)	(50.09,5.72)
			(48.92,17.35)	(50.08,6.00)
		PAPUF	(50.08,30.14)	(50.38,17.69)
	512	APUF	(50.03,1.23)	(50.00,1.49)
		DAPUF	(47.26,20.08)	(50.01,8.02)
			(47.36,19.81)	(50.02,7.81)
		PAPUF	(42.84,29.15)	(49.30,17.56)
FPGA	64	APUF	(54.74,1.12)	(3.25,0.27)
		DAPUF	(99.60,0.87)	(0.79,1.01)
			(98.52,1.77)	(2.85,1.94)
		PAPUF	(0,0)	(0,0)
	128	APUF	(51.96,1.49)	(3.72,0.53)
		DAPUF	(88.32,8.49)	(12.09,4.93)
			(24.53,18.42)	(24.56,12.19)
		PAPUF	(0,0)	(0,0)

Note: DAPUF has two response bits and hence we have provided two entries for quality metrics.



(a) 64 Bit (b) 512 Bit

Fig. 8: Comparison of uniformity of simulated APUF, DAPUF, and PAPUF with 64-bit and 512-bit challenge. Difference between the uniformity properties of 512-bit PAPUF and DAPUF is reduced with respect to 64-bit variants. It seems that they would have similar uniformity properties for large challenge length.

For the simulations, we created 100 instances of each PUF design, and the uniformity and uniqueness metrics were computed based on these population of PUF instances using 10,000 CRPs. Results of simulated PUF designs are summarized in Table II. Figure 8 delineates the uniformity comparison with respect to 64-bit and 512-bit simulated PUF instances, and it can be observed that number of APUF instances with good uniformity increases with the increasing challenge length, whereas numbers of PAPUF and DAPUF instances with good uniformity decrease with increasing challenge size.

TABLE III: Reliability of APUF, PAPUF and DAPUF on five Artix-7 FPGAs

Size	PUF	Reliability (%)				
		FPGA-1	FPGA-2	FPGA-3	FPGA-4	FPGA-5
64	APUF	99.78	99.76	99.82	99.76	99.75
	DAPUF	99.38	99.99	100.00	100.00	100.00
		99.09	99.97	99.86	99.76	99.58
	PAPUF	100.00	100.00	100.00	100.00	100.00
128	APUF	99.76	99.63	99.76	99.61	99.67
	DAPUF	98.92	99.77	99.67	98.64	99.20
		99.14	98.28	98.67	98.80	99.13
	PAPUF	100.00	100.00	100.00	100.00	100.00

Note: DAPUF has two response bits and hence we have provided two entries for reliability.

TABLE IV: Fraction of "good" PUF instances (G) in the populations of 100 instances of simulated APUF, PAPUF and DAPUF designs

Size	PUF	No. of Good PUF Instances (%)			
		$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.15$	$\epsilon = 0.20$
64	APUF	83	100	100	100
	DAPUF	19	39	52	65
	PAPUF	6	18	26	43
128	APUF	96	100	100	100
	DAPUF	21	37	50	58
	PAPUF	9	20	34	43
256	APUF	99	100	100	100
	DAPUF	12	25	46	62
	PAPUF	10	22	35	42
512	APUF	100	100	100	100
	DAPUF	21	33	56	68
	PAPUF	10	18	29	36

Note: The ϵ is the bias tolerance parameter as defined in Section V.

We also implemented the same PUF designs on five *Xilinx* Artix-7 (XC7A100T) FPGAs (for challenge sizes of 64-bit and 128-bit). The golden response for each FPGA PUF instance was estimated by employing majority voting of CRPs collected over 11 different time instants at normal operating condition (i.e., at normal room temperature and with standard supply voltage). For the PAPUF implementation on FPGA, we excluded the use of tuning elements (that was exploited to avoid implementation bias in [20], [25]) to accentuate the impact of architectural bias. We have implemented PAPUF by using *hard macro* (HM) to instantiate the top and bottom paths with reduced implementation bias. Uniformity and uniqueness metrics of FPGA implemented APUF, PAPUF and DAPUF are reported in Table II. From these results, it is evident that the FPGA-implemented PAPUF instances (without tuning elements) have extremely poor quality, so much so that they can hardly be termed as PUF instances. Since DAPUF can be thought to have an architecture which is the combination of those of the APUF and the PAPUF, its delay difference distribution follows normal distribution with large variance compared to PAPUF. Hence, DAPUF has better uniformity and uniqueness compared to PAPUF, but reliability of the DAPUF is poor compared to the PAPUF, as reported in Table III. Following fact can be observed by comparing simulation and FPGA results: Difference between the qualities of PAPUF and DAPUF is gradually reducing with the increasing challenge length for simulation, but this trend is not clearly observed in FPGA implementation. One reason can be the effects of other components in system are comparatively more on longer PUF.

In Table IV, we have computed the number of good PUF

instances (G) in the populations of 100 instances. From this result, it can be observed that, for $\epsilon = 0.1$, average values of G over different challenge sizes for APUF, DAPUF and PAPUF are 100%, 33.5% and 19.5%, respectively, and it matches the theory in Section V. Since we have only five FPGA implemented PUF instances for each design, we exclude this computation for FPGA implementation.

Although PAPUF and DAPUF were proposed as FPGA friendly variants of APUFs, actually they are inferior to APUF (in the absence of tuning elements), except with respect to the reliability metric. From the point of view of implementation, DAPUF is a superior than PAPUF because it does not need to modify the switching logic of APUF, and an APUF design can easily be modified to the DAPUF structure by instantiating two instances of the APUF design. It is worth mentioning that from the traditional machine learning based modeling attack perspective, all the three variants are vulnerable to modeling attack.

IX. Conclusion

In this work, we have introduced the concept of *architectural bias* to compare the architectures of APUF, PAPUF and DAPUF. We have developed the linear additive delay models of PAPUF and DAPUF to estimate the bias in their challenge-response behavior due to the weakness in their architectures. A comparative study of APUF and its two variants – PAPUF and DAPUF has also been presented with respect to their architectural bias. It is observed, theoretically, that for bias tolerance value $\epsilon = 0.1$, 99.7% instances of a APUF design are good, while number of good PUF instances for DAPUF and PAPUF are 33.5% and 20%. Our reported results also follow this fact. We have also developed a scheme to measure a given pair of randomly selected instances of two different PUF designs. To summarize, we have shown that none of the APUF design variants, ideally, are superior than APUF itself, except the fact that they are easier to design on FPGA in their ideal form. From the uniformity and uniqueness aspect, DAPUF is a better alternative of APUF on FPGA than PAPUF. However, due to the implementation induced bias on FPGA, APUF design fails to exhibit its superiority with respect to uniqueness. From this study, it is evident that architecture of delay PUF design has a significant impact on its performance; only easy-to-implement PUF architecture is not sufficient to be considered as good PUF design, for example PAPUF. As a future work, we try to apply the notion of architectural bias on other architectures of strong delay PUFs.

References

[1] R. S. Pappu, "Physical one-way functions," Ph.D. dissertation, Massachusetts Institute of Technology, March 2001.

[2] D. Lim, "Extracting Secret Keys from Integrated Circuits," Master's thesis, MIT, USA, 2004.

[3] R. Maes and I. Verbauwhede, "Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions," in *Towards Hardware-Intrinsic Security*, ser. Information Security and Cryptography, A.-R. Sadeghi and D. Naccache, Eds. Berlin Heidelberg: Springer, 2010, pp. 3–37.

[4] S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "Extended abstract: The butterfly PUF protecting IP on every FPGA," in *Proc. of IEEE HOST*, June 2008, pp. 67–70.

[5] M.-D. M. Yu, D. M'Raïhi, R. Sowell, and S. Devadas, "Lightweight and Secure PUF Key Storage Using Limits of Machine Learning," in *Proc. of 13th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)* , vol. 6917. Springer Berlin / Heidelberg, 2011, pp. 358–373.

[6] C. Brzuska, M. Fischlin, H. Schrder, and S. Katzenbeisser, "Physically Uncloneable Functions in the Universal Composition Framework," in *Advances in Cryptology (CRYPTO)*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed. Springer Berlin / Heidelberg, 2011, vol. 6841, pp. 51–70.

[7] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls, "Memory Leakage-Resilient Encryption Based on Physically Unclonable Functions," in *Proc. of ASIACRYPT* . Berlin, Heidelberg: Springer-Verlag, 2009, pp. 685–702.

[8] 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.

[9] J. Tobisch and G. T. Becker, "On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation," in *Proc. of 11th International Workshop on RFIDsec*, 2015, pp. 17–31.

[10] G. 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.

[11] D. P. Sahoo, P. H. Nguyen, D. Mukhopadhyay, and R. S. Chakraborty, "A Case of Lightweight PUF Constructions: Cryptanalysis and Machine Learning Attacks," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2015.

[12] R. Kumar and W. Burleson, "Side-Channel Assisted Modeling Attacks on Feed-Forward Arbiter PUFs Using Silicon Data," in *Proc. of 11th International Workshop on RFIDsec*, 2015, pp. 53–67.

[13] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUFs," in *Proc. of the 2008 IEEE/ACM International Conference on Computer-Aided Design(ICCAD)*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 670–673.

[14] D. P. Sahoo, D. Mukhopadhyay, and R. S. Chakraborty, "Design of Low Area-overhead Ring Oscillator PUF with Large Challenge Space," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2013.

[15] D. P. Sahoo, S. Saha, D. Mukhopadhyay, R. S. Chakraborty, and H. Kapoor, "Composite PUF: A New Design Paradigm for Physically Unclonable Functions on FPGA," in *Proc. of IEEE HOST*, Arlington, VA, USA, May 2014, pp. 50–55.

[16] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar, and W. 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)*, 2014, pp. 476–492.

[17] 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.

[18] Y. Hori, T. Yoshida, T. Katashita, and A. Satoh, "Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs," in *Proc. of IEEE ReConFig*, dec. 2010, pp. 298 –303.

[19] A. Maiti, V. Gunreddy, and P. Schaumont, "A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions," *IACR Cryptology ePrint Archive*, vol. 2011, p. 657, 2011.

[20] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines," in *IEEE International Workshop on Information Forensics and Security (WIFS)*, Dec 2010, pp. 1–6.

[21] T. Machida, D. Yamamoto, M. Iwamoto, and K. 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)*, 2014, pp. 871–878.

[22] ——, "Implementation of double arbiter PUF and its performance evaluation on FPGA," in *Proc. of the 20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015, pp. 6–7.

[23] Y. Lao and K. K. Parhi, "Statistical Analysis of MUX-Based Physical Unclonable Functions," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 33, no. 5, pp. 649–662, 2014.

[24] Z. C. Jouini, J.-L. DANGER, and L. Bossuet, "Performance Evaluation of Physically Unclonable Function by Delay Statistics," in *Proc. of 9th IEEE International NEWCAS conference*, 2011.

[25] M. Majzoobi, A. Kharaya, F. Koushanfar, and S. Devadas, "Automated Design, Implementation, and Evaluation of Arbiter-based PUF on FPGA using Programmable Delay Lines," *IACR Cryptology ePrint Archive*, vol. 2014, p. 639, 2014. [Online]. Available: http://eprint.iacr.org/2014/639