On the Architectural Analysis of Arbiter Delay PUF Variants[*]

DURGA PRASAD SAHOO, PHUONG HA NGUYEN, RAJAT SUBHRA CHAKRABORTY, and DEBDEEP MUKHOPADHYAY, Indian Institute of Technology Kharagpur, India

The *Arbiter Physically Unclonable Function* (APUF) is a widely used strong delay PUF design. There are two FPGA variants of this design, namely, Programmable Delay Line APUF (PAPUF) and Double APUF (DAPUF) to mitigate the FPGA platform specific implementation issues. In this paper, we introduce the idea of *Architectural Bias* to compare the impact of the architecture of these APUF designs on their design bias. The biased challenge-response behavior of a delay PUF implies the nonuniform distributions of 0's and 1's in its response, and if this bias is due to the architectural issue of the PUF design, then it is called "Architectural Bias". Another important source of bias is the implementation issue specific to an implementation platform. According to our study, a good PUF architecture results in PUF instances with small amount of architectural bias. In this paper, we provide a comparison of APUF, PAPUF and DAPUF based on their architectural bias values. In addition, we also compare these APUF architectures with respect to fulfilling the Strict Avalanche Criterion (SAC) and robustness against the machine learning (ML) based modeling attack. We validate our theoretical findings with Matlab based simulations, and the results reveal that the classic APUF has the least architectural bias, followed by DAPUF and PAPUF, respectively. We also conclude from the experimental results that the SAC property of DAPUF is better than that of APUF and PAPUF, and PAPUF's SAC property is significantly poor. However, our analyses indicate that these APUF variants are vulnerable to ML-based modeling attack.

Additional Key Words and Phrases: Architectural bias, arbiter PUF (APUF), double APUF (DAPUF), physically unclonable function (PUF), programmable delay line (PDL), PDL based APUF (PAPUF), modeling attack, strict avalanche criterion.

1 INTRODUCTION

Implementation of Arbiter Physically Unclonable Function (APUF) [5] on FPGA platform is a challenging task due to predefined logic array layout, and relative lack of designer's control on routing offered by existing FPGA CAD tools [12]. Consequently, implementation induced bias (i.e. non-uniform distribution of 0's and 1's in response bit-string) creeps into the APUFs implemented on FPGAs, and results in APUF implementation with relatively poor statistical properties [2]. To overcome the shortcomings, researchers came up with two APUF design variants specifically targeting FPGA devices: *Programmable Delay Line* (PDL) based APUF (PAPUF) [12] and *Double Arbiter PUF* (DAPUF) [6, 7]. The design objective of PAPUF and DAPUF is to mitigate the implementation induced bias specific to the FPGA platform. However, there is no study in the literature that systematically compares the pros and cons of these APUF variants. This motivates us to reinvestigate these delay PUF architectures in the search of answers to the following queries:

(1) Are the performance metrics (e.g. uniformity, uniqueness and reliability) of PAPUF and DAPUF upto the level of APUF, while we consider simulation of these architectures?
(2) How the PAPUF architecture is different from DAPUF architecture with respect to performance metrics?
(3) How are the security properties of these APUF architectures different?

[*]This is an updated version of the article "Architectural Bias: a Novel Statistical Metric to Evaluate Arbiter PUF Variants."

Authors' address: Durga Prasad Sahoo; Phuong Ha Nguyen; Rajat Subhra Chakraborty; Debdeep Mukhopadhyay, Indian Institute of Technology Kharagpur, Secured Embedded Architecture Laboratory (SEAL), Computer Science and Engineering, Kharagpur, West Bengal, 721302, India, Email:dpsahoo.cs@gmail.com,phuongha.ntu@gmail.com,rschakraborty@cse.iitkgp.ernet.in,debdeep@cse.iitkgp.ernet.in.

Note that APUF and its variants are used as the primitive component in various secure strong PUF designs (e.g., XOR PUF [21], LSPUF [13] and MPUF [20]) as well as security protocols (e.g. Slender PUF protocol [15] and Lockdown authentication protocol [22]). Thus, a proper guideline for selecting an APUF architecture for a specific platform would helpful for system developer. This is another motivation of focusing on the APUF architectures.

In this paper, we present a platform-independent study of these architectures to understand their ideal behavior, and how they are supposed to work in practice. In this context, we introduce the notion of *Architectural Bias*, which is a bias (i.e., non-uniform distribution of 0's and 1's) in challenge-response behavior of the APUF variants due to their architectures. This would be the basis for comparing the performance metrics of APUF and its variants.

Modeling attack becomes a common security vulnerability for strong delay PUF architectures, and there is no exception for these APUF designs. Modeling attack on APUF is known from very early days of this design [5], and the parity vector of challenge is used as feature set for machine learning algorithm. However, there is no reported result on feature set formats for PAPUF and DAPUF. To make a fair comparison of modeling vulnerability of these APUF designs, we develop linear additive delay models for PAPUF and DAPUF. These model can give the right format of feature sets which are required for efficient machine learning based modeling attack. Besides the modeling attack, we also compare the APUF variants with respect to their vulnerability to statistical attacks based on their Strict Avalanche Criterion (SCA) [17].

Our contributions in this work are summarized as follows:

(1) We develop analytic delay models for the DAPUF and PAPUF circuits, as these form the foundation of the analysis in the rest of the paper. These models can be considered as the extension of the APUF model developed in [5].
(2) We introduce the notion of *architectural bias* based on the generic behavior of strong delay PUF. This bias exists in the design regardless of how good implementation is done on any platform like FPGA or ASIC, i.e., bias is platform independent.
(3) We propose an evaluation scheme for delay PUF designs based on their architectural bias values. This scheme estimates the fraction of good instances (which we called *goodness factor*) of a delay PUF design for a given bias tolerance limit ϵ, either using the delay model of the PUF design or based on the challenge-response dataset of PUF instances. The goodness of the architectures of a pair of PUF designs can be decided by comparing the values of the corresponding goodness factors.
(4) As the case studies, we have applied the notion of architectural bias to compare APUF, PAPUF and DAPUF. This study shows that classic APUF architecture has less architectural bias compared to both the PAPUF and DAPUF. It is also shown that DAPUF has comparatively less architecture bias than PAPUF. To the best of our knowledge, this is first work that provide a comparison of APUF, PAPUF and DAPUF based on their architectures. Based on the analytical models of APUF, PAPUF and DAPUF, we quantitatively estimate their goodness factors. We show that for bias tolerance parameter $\epsilon = 0.1$, approximately 99% of APUF instances are good, while this quantity is 33% for DAPUF and 20% for PAPUF.
(5) The impacts of architectural bias on the PUF performance metrics like uniformity, uniqueness and reliability are discussed in brief. Interestingly, we demonstrate that large architectural bias results in poor uniformity and uniqueness, but improved reliability.
(6) We also discuss the modeling attack, and statistical attack based on the SAC properties of APUF and its variants.
(7) Finally, we realize the APUF, PAPUF and DAPUF designs by using *Matlab* based simulation to validate the notion of architectural bias, modeling attacks and SAC property.

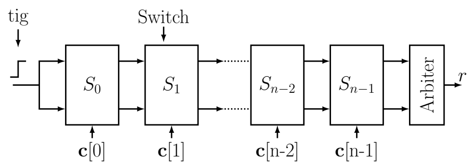


Fig. 1. An n-stage APUF. S_i denotes the i-th switching stage.

The rest of the paper is organized as follows. In Section 2.1, we introduce a notation system that will be used throughout the paper, and then develop linear additive models of PAPUF and DAPUF in Section 2.2. The notion of bias in PUF is explained in Section 3. The notion of architectural bias and evaluation of delay PUF designs using it are discussed in Sections 4 and 5, respectively. A comparison of APUF variants based on their architectural bias is presented in Section 6. In Section 7, the impacts of architectural bias on uniqueness and reliability properties are discussed. Security analysis of APUF variants are provided in Section 8 using SAC property and machine learning based modeling attack. We validate our findings based on the simulated APUF designs in Section 9. A discussion on APUF, PAPUF and DAPUF architectures is provided in Section 10. Finally, the paper is concluded in Section 11.

2 NOTATION AND DELAY MODELING

2.1 Notations

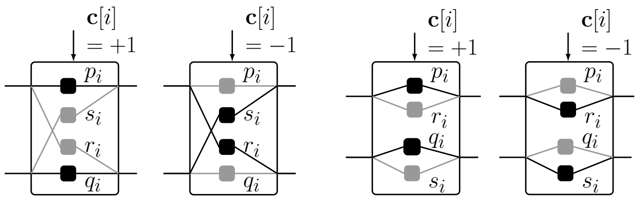
The following notation system will be used in the rest of the paper. A vector is represented by lowercase letter in bold font, e.g $\mathbf{a}$. A vector with m-components is represented as: $\mathbf{a} = (\mathbf{a}[0], \ldots, \mathbf{a}[i], \ldots, \mathbf{a}[m-1])$, where $\mathbf{a}[i]$ is the $(i+1)$-th 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. 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 $\mu_{\rm X}$, $\sigma^2_{\rm X}$ are used to denote mean and variance of the random variable X, respectively. A probability distribution is denoted by $\mathcal{D}$. The notation $X \sim \mathcal{D}_X$ implies that the random variable X follows the probability distribution $\mathcal{D}_X$. Random variable X following a Gaussian probability distribution function is denoted as $X \sim \mathcal{N}(\mu, \sigma^2)$. The $\phi_{\mu,\sigma^2}()$ 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.

2.2 Linear Additive Delay Models of APUF and Its Variants

In this section, we provide the compact representation of APUF delay model, and derive delay models for PAPUF and DAPUF. *Note that during the derivation of these models, challenges to the PUF follow a bipolar encoding, i.e. logic-0 is encoded as '+1' and logic-1 is encoded as '-1'.* In these models, our objective is only to model the delay differences after the last switching stage, without the consideration of arbiter circuit (cf. Fig. 1).

2.2.1 Model of APUF. Let us briefly recap the linear additive delay model of APUF (cf. Fig. 1) as derived in [5]. Let p_i, r_i, s_i and q_i be the four delay components of the i-th switching stage of APUF as shown in Fig. 2a. The delay difference of top and bottom paths of an APUF instance can be modeled as:

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



(a) APUF's i-th stage (b) PAPUF's i-th stage

Fig. 2. Switching stages of classic APUF and PAPUF. Depending on the challenge bit $\mathbf{c}[i] \in \{+1, -1\}$, a pair of delay elements is selected.

where the feature vector[1] $\mathbf{\Phi}$ is the parity vector of challenge $\mathbf{c}$ defined as:

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

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

$$\mathbf{w}[0] = \alpha_0, \qquad \mathbf{w}[n] = \beta_{n-1}, \qquad \text{and} \qquad \mathbf{w}[i] = \alpha_i + \beta_{i-1} \quad \text{for } i = 1, \ldots, n-1, \quad \text{where} \tag{3}$$

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

Hence, $\Delta(n-1) = \langle \mathbf{w}, \mathbf{\Phi} \rangle$ represents a linear additive delay model of an APUF. Note that α_i and β_i notations are used only to make representation more compact and these qualities are defined based on the delay components of APUF switches.

2.2.2 Model of PAPUF. The principal difference between APUF and PAPUF [12] lies in the design of switching stages. Fig. 2 shows the structural difference between the switching elements of APUF and PAPUF. In contrast to APUF which uses path-swapping switches, PAPUF uses non-swapping path based switches to reduce the implementation bias on FPGA. In the PAPUF design in [12], the authors introduced additional delay tuning elements to mitigate the bias. However, we exclude the tuning part in model building of PAPUF to aid us in analyzing the architecture-specific bias. The tuning part was proposed to tune the bias, but here, we are interested in analyzing the factor of this bias. Next, we develop the delay model of PAPUF.

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.

Let p_{i+1} and r_{i+1} be two delay components in the top part of $(i+1)$-th stage, and q_{i+1} and s_{i+1} be the other two delay components in the bottom part of $(i+1)$-th stage (cf. Fig. 2b). 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:

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

[1]The symbol Φ in bold font (i.e., $\mathbf{\Phi}$) represents a vector, whereas $\Phi()$ denotes the CDF of Normal distribution.

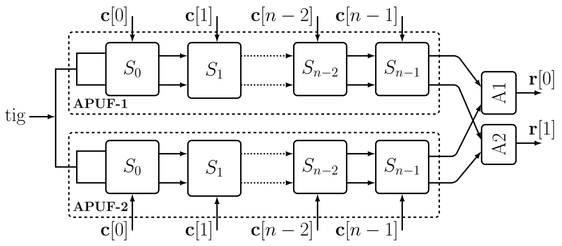


Fig. 3. An n-stage DAPUF with challenge **c**. A1 and A2 are two arbiter circuits.

Then delay difference between top and bottom paths can be estimated as:

$$\Delta(i+1) = \delta_t(i+1) - \delta_b(i+1) = \frac{1+\mathbf{c}[i+1]}{2}(\Delta(i) + p_{i+1} - q_{i+1}) \tag{4}$$
$$+\frac{1-\mathbf{c}[i+1]}{2}(\Delta(i) + r_{i+1} - s_{i+1}) = \Delta(i) + \alpha_{i+1}\mathbf{c}[i+1] + \beta_{i+1}, \qquad \text{where}$$
$$\alpha_{i+1} = \frac{p_{i+1} - r_{i+1} - q_{i+1} + s_{i+1}}{2}, \text{ and } \beta_{i+1} = \frac{p_{i+1} + r_{i+1} - q_{i+1} - s_{i+1}}{2}.$$

Let us assume that $\Delta(-1) = 0$, and then the delay difference $\Delta(n-1)$ after the final stage can be:

$$\Delta(n-1) = \sum_{i=0}^{n-1} \alpha_i \mathbf{c}[i] + \sum_{i=0}^{n-1} \beta_i = \sum_{i=0}^{n} \mathbf{w}[i]\mathbf{\Phi}[i] = \langle \mathbf{w}, \mathbf{\Phi} \rangle, \quad \text{where} \tag{5}$$
$$\mathbf{w}[n] = (\beta_{n-1} + \cdots + \beta_0) \qquad \text{and} \qquad \mathbf{w}[i] = \alpha_i \quad \text{for } i = 0, \ldots, n-1,$$
$$\mathbf{\Phi}[0:n-1] = \mathbf{c}[0:n-1], \qquad \text{and} \qquad \mathbf{\Phi}[n] = 1.$$

Thus, PAPUF also follows a linear additive delay model, very similar to the classic APUF [5]. However, there are also important differences between the delay models of classic APUF and PAPUF:

(1) The feature vector $\mathbf{\Phi}$ in PAPUF model is the challenge **c** itself, while $\mathbf{\Phi}$ in APUF is the parity vector of challenge **c**.
(2) All elements of **w** (except $\mathbf{w}[n]$) in PAPUF model depend only on the α values, whereas most of the elements of **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 $\beta_0, \cdots, \beta_{n-1}$. In both the cases, α_i and β_i have different definitions. Later, this observation will be used to compare the architectural bias values of APUF and PAPUF instances.

2.2.3 Model of DAPUF. As shown in Fig. 3, the DAPUF [6] consists of two APUF instances, which are evaluated in parallel for a given challenge, and two 1-bit responses **r**[0] and **r**[0] are generated. The upper paths of both APUF instances are connected to an arbiter circuit (A1) to compare their delays. Similarly, signal propagation delays of both the lower paths are compared using another arbiter circuit (A2). The main assumption made for this design scheme is that FPGA CAD tool maintains similar routing for 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 to be quasi-identical using *hard macros* (excluding the arbiters), the designer can expect elimination of the implementation bias.

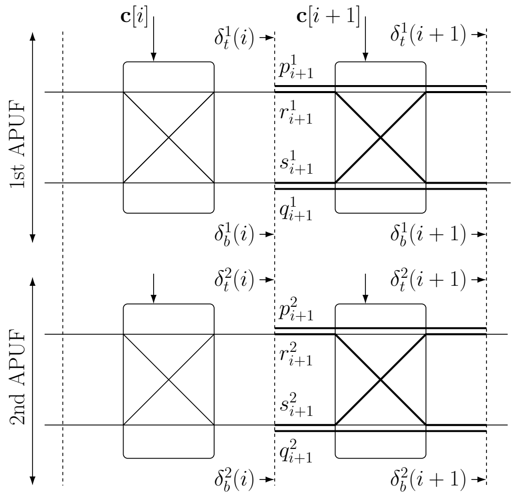


Fig. 4. The i-th and $(i+1)$-th switching stage of DAPUF.

Let $\Delta_t(n-1)$, $\Delta_b(n-1)$ be the delay differences of two top paths and two bottom paths of DAPUF, respectively. Then, the two responses of DAPUF are defined as:

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

Now, we develop an analytical model for $\Delta_t(n-1)$, and the model for $\Delta_b(n-1)$ can be derived in a similar way. 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 i-th switch of j-th APUF, respectively. Let p_{i+1}^j, q_{i+1}^j, r_{i+1}^j and s_{i+1}^j be the values of four delay components of the $(i+1)$-th switching stage of j-th APUF, as shown in Fig. 4. Now, we can derive the delay of trigger signal at the output of $(i+1)$-th stage with challenge $\mathbf{c} \in \{+1, -1\}$ as follows:

$$\begin{aligned}
\delta_t^1(i+1) &= \frac{(1-\mathbf{c}[i+1])}{2}\left[\delta_t^1(i) + p_{i+1}^1\right] + \frac{(1+\mathbf{c}[i+1])}{2}\left[\delta_b^1(i) + s_{i+1}^1\right] \\
\delta_t^2(i+1) &= \frac{(1-\mathbf{c}[i+1])}{2}\left[\delta_t^2(i) + p_{i+1}^2\right] + \frac{(1+\mathbf{c}[i+1])}{2}\left[\delta_b^2(i) + s_{i+1}^2\right] \\
\delta_b^1(i+1) &= \frac{(1-\mathbf{c}[i+1])}{2}\left[\delta_b^1(i) + q_{i+1}^1\right] + \frac{(1+\mathbf{c}[i+1])}{2}\left[\delta_t^1(i) + r_{i+1}^1\right] \\
\delta_b^2(i+1) &= \frac{(1-\mathbf{c}[i+1])}{2}\left[\delta_b^2(i) + q_{i+1}^2\right] + \frac{(1+\mathbf{c}[i+1])}{2}\left[\delta_t^2(i) + r_{i+1}^2\right]
\end{aligned} \tag{6}$$

In case of DAPUF, we compute delay difference of top paths of two APUF instances to generate response $\mathbf{r}[0]$. Similarly, $\mathbf{r}[1]$ is generated by computing 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 between top paths of two APUF instances after the i-th and $(i+1)$-th stages of DAPUF, respectively. Similarly, let $\Delta_b(i)$ and $\Delta_b(i+1)$ represent the delay differences between bottom paths of two APUF instances after the i-th and $(i+1)$-th stages, respectively. Then, we have:

$$\Delta_t(i+1) = \delta_t^1(i+1) - \delta_t^2(i+1) = \Delta_t(i)\frac{(1-\mathbf{c}[i+1])}{2} + \Delta_b(i)\frac{(1+\mathbf{c}[i+1])}{2} + \alpha_{i+1}^t\mathbf{c}[i+1] + \beta_{i+1}^t,$$

$$\text{where} \quad \alpha_{i+1}^t = \frac{p_{i+1}^2 - p_{i+1}^1 + s_{i+1}^1 - s_{i+1}^2}{2} \text{ and } \beta_{i+1}^t = \frac{p_{i+1}^1 - p_{i+1}^2 + s_{i+1}^1 - s_{i+1}^2}{2}$$

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

$$\text{where} \quad \alpha_{i+1}^b = \frac{q_{i+1}^2 - q_{i+1}^1 + r_{i+1}^1 - r_{i+1}^2}{2} \text{ and } \beta_{i+1}^b = \frac{q_{i+1}^1 - q_{i+1}^2 + r_{i+1}^1 - r_{i+1}^2}{2}.$$

Let us assume that $\Delta_t(-1) = 0$ and $\Delta_b(-1) = 0$, and then $\Delta_t(n-1)$ can be represented as:

$$\Delta_t(n-1) = \langle \mathbf{x}_t, \mathbf{c} \rangle + \langle \mathbf{y}_t, \mathbf{p} \rangle = \langle \mathbf{w}, \mathbf{\Phi} \rangle, \tag{7}$$

where $2n$-bit weight vector $\mathbf{w} = (\mathbf{x}_t, \mathbf{y}_t)$ and $2n$-bit feature vector $\mathbf{\Phi} = (\mathbf{c}, \mathbf{p})$. Similarly, we can compute $\Delta_b(n-1)$ of DAPUF, the delay difference of two bottom paths. Let us define an n-bit vector $\mathbf{p}$ based on challenge $\mathbf{c}$ as:

$$\mathbf{p}[n-1] = 1, \qquad \text{and} \qquad \mathbf{p}[i] = \prod_{j=i}^{n-1} \mathbf{c}[i] \quad \text{for } 0 \le i \le n-2 \tag{8}$$

The n-bit vectors $\mathbf{x}$ and $\mathbf{y}$ are defined as follows:

$$\mathbf{x}_t[i] = \begin{cases} \frac{\alpha_i^t + \alpha_i^b}{2} & \text{if } 0 \le i < n-1 \\ \frac{\beta_b^{i-1} - \beta_t^{i-1} + 2\alpha_i^t}{2} & \text{if } i = n-1 \end{cases} \qquad \mathbf{y}_t[0] = \begin{cases} \frac{\alpha_0^b - \alpha_0^t}{2} & \text{if } n \text{ is even} \\ \frac{\alpha_0^t - \alpha_0^b}{2} & \text{if } n \text{ is odd} \end{cases}$$

For $1 \le i \le n-2$,

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

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

In the definitions of $\mathbf{x}_t$ and $\mathbf{y}_t$, we have used α_i^t and β_i^t values to make the definitions more comprehensive, and these quantities are defined based on the delay parameters of delay components of APUF switches. In case of DAPUF, we have two APUFs, and four delay components of the $(i+1)$-th switching stage of j-th APUF for $j \in \{1, 2\}$ are denoted by p_{i+1}^j, q_{i+1}^j, r_{i+1}^j and s_{i+1}^j, as shown in Fig. 4. Then, α_{i+1}^t and β_{i+1}^t are defined as follows:

$$\alpha_{i+1}^t = \frac{p_{i+1}^2 - p_{i+1}^1 + s_{i+1}^1 - s_{i+1}^2}{2}, \quad \beta_{i+1}^t = \frac{p_{i+1}^1 - p_{i+1}^2 + s_{i+1}^1 - s_{i+1}^2}{2}, \tag{9}$$
$$\alpha_{i+1}^b = \frac{q_{i+1}^2 - q_{i+1}^1 + r_{i+1}^1 - r_{i+1}^2}{2}, \quad \beta_{i+1}^b = \frac{q_{i+1}^1 - q_{i+1}^2 + r_{i+1}^1 - r_{i+1}^2}{2}.$$

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 challenge and parity vectors. Thus, we can say that the delay model of a DAPUF is a combination of delay models of classical APUF and a PAPUF.

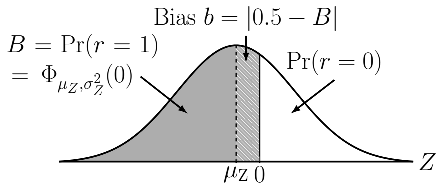


Fig. 5. An example delineating bias in the behavior of a delay PUF instance. In this case, response generated by a PUF instance is biased towards '1'.

Next we explain the bias in the delay PUF and what are the possible factors for these bias.

3 BIAS IN CHALLENGE-RESPONSE BEHAVIOR OF DEALY PUF

In general, for delay PUFs (e.g. APUF [5]), the response to a given challenge $\mathbf{c}$ is determined by the delay difference $z_{\mathbf{c}}$ between a pair of paths established by challenge $\mathbf{c}$ as:

$$r = \begin{cases} 1 & \text{if } z_{\mathbf{c}} < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

In case of nanoscale CMOS circuits, delay variation in circuit due the random process variations follows a Gaussian distribution [9], and this fact has been used in the past to analyze delay PUFs [4, 5]. Let $Z \sim \mathcal{N}(\mu_Z, \sigma_Z^2)$ be a Gaussian random variable representing the delay differences of a delay PUF instance, whereas $z_{\mathbf{c}}$ represents the delay difference for a particular challenge $\mathbf{c}$, i.e., $Z = z_{\mathbf{c}}$ for challenge $\mathbf{c}$. It is desirable that an ideal delay PUF should generate responses where 0's and 1's are equiprobable. The uniformity metric of a PUF instance estimate the fractions of 0's and 1's in PUF responses, and it is defined based on the delay difference distribution[2] as [5]:

$$B = \Pr(r = 1) = \Pr(Z \le 0) = \frac{1}{\sigma_Z\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{(u-\mu_Z)^2}{2\sigma_Z^2}} \, \mathrm{d}u = \Phi_{\mu_Z,\sigma_Z^2}(0) = \Phi\left(\frac{-\mu_Z}{\sigma_Z}\right). \tag{11}$$

Equation (11) is another way of stating that the ideal value of PUF uniformity is $B = \Phi(0) = 0.5$ when $(\frac{-\mu_Z}{\sigma_Z}) = 0$. If a PUF design fails to achieve $(\frac{-\mu_Z}{\sigma_Z}) \neq 0$, then its challenge-response behavior exhibits a *bias*. The bias value can be estimated as:

$$b = |\Phi(0) - B| = |0.5 - B|\,. \tag{12}$$

Figure 5 shows the behavior of a delay PUF instance with $\mu_Z < 0$, and it results a bias in the behavior of PUF instance towards '1', as the response is defined by Eq. (10). Two key reasons for this bias can be:

(1) **Imperfect Implementation.** The most important requirement while designing a delay PUF 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 comparatively difficult to achieve for PUFs implemented in FPGA platform compared to ASIC, due to predefined gate-array layout of FPGA and less control on the routing provided by FPGA CAD tool. Thus, non-ideal implementations of delay PUFs can result in bias. We term this implementation induced bias as *"implementation bias"*.

[2]Here, $\Phi(\cdot)$ and $\Phi_{\mu,\sigma^2}(\cdot)$ are CDFs of standard normal and normal random variables, respectively.

(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"*.

Next, we focus on the architectural bias, which is the main topic of discussion in this paper. Since the architectural bias is not related to any implementation platform, in this paper, we use simulated PUFs to estimate their architectural bias.

4 ARCHITECTURAL BIAS IN DELAY PUF

By architectural bias, we refer to the bias in challenge-response behavior of a delay PUF design that is *instance-specific, challenge invariant and independent of the implementation platform.* Let us elaborate the "instance-specific" and "challenge invariant" properties of the architectural bias. The reader might think how an architectural issue can results in a unique bias value for each PUF instance following the same architecture. The reason is that the delay elements of a PUF instances are affected by the random process variations, and delay values follow a Gaussian distribution with zero mean. In the context of PUF, "instance-specific" property of a metric implies that the value of this metric follows an Gaussian distribution. On the other hand, "challenge invariant" properties of architectural bias implies that this bias would be present as a fixed additive offset to the delay difference of each challenge of a given PUF instance. Note that implementation bias does not have this "challenge invariant" property. Next, we formalize the notion of architectural bias.

Let Z be a random variable representing the delay differences of a delay PUF instance due to the application of challenges, and it follows a Gaussian distribution $\mathcal{D}_Z = \mathcal{N}(\mu_Z, \sigma_Z^2)$. To explain the notion of architectural bias more precisely, we express random variable Z as follows:

$$Z = X + y, \tag{13}$$

where $X \sim \mathcal{D}_X = \mathcal{N}(0, \sigma_X^2)$, $y \leftarrow \mathcal{D}_y = \mathcal{N}(0, \sigma_y^2)$, $\mu_Z = y$ and $\sigma_Z^2 = \sigma_X^2$. Here, X is a Gaussian random variable, and $\mathcal{D}_X$ and $\mathcal{D}_y$ are two Gaussian distributions. The notation "$y \leftarrow \mathcal{D}_y$" implies that y is sampled from distribution $\mathcal{D}_y$.

The key difference between X and y is that while *values of X depend on challenge and process variations of the device where the PUF is embedded, y only depends on process variations of the device.* Since y is independent of challenge, it is a constant for a given PUF instance and appears as a fixed additive offset in delay difference of each challenge. As a consequence, y value introduces a shift to the mean value of delay distribution $\mathcal{D}_Z$ (i.e. $\mu_Z \neq 0$) from its ideal value $\mu_Z = 0$, which makes the distributions of 0's and 1's in PUF responses non-uniform (cf. Fig. 5). Let us call this bias in the distributions of 0's and 1's as *"Architectural Bias"*. For a given PUF instance, it is estimated as:

$$b_{\text{archi}} = |0.5 - B|,$$

where $B = \Phi(\frac{-y}{\sigma_Z})$ is the uniformity metric as defined in Eq. (11). *It is desirable that a good PUF design should have architecture induced bias close to zero (i.e. $b_{\text{archi}} \approx 0$), for most of its instances in a given population.*

Now, let us consider a population $\mathcal{I} = \{I_1, \ldots, I_{K-1}\}$ of instances of a PUF design P. Each instance $I_i, i \in [0 : K-1]$ is associated with a 2-tuple (X_i, y_i), where $X_i \sim \mathcal{D}_X$ and $y_i \leftarrow \mathcal{D}_y$. The (X_i, y_i) defines the random variable $Z_i = X_i + y_i$ (cf. Eq. (13)) corresponding to instance I_i. For all instances of P, X_i's are independent and identically distributed random variables, as delay parameters of PUF instances are sampled independently from a common Gaussian distribution and they have the same challenge space. Thus, $Z_i, i \in [0, K-1]$, follows a Gaussian distribution $\mathcal{D}_{Z_i} = \mathcal{N}(y_i, \sigma_Z^2)$, where $\sigma_Z^2 = \sigma_X^2$. Since y_i's are independently sampled from a Gaussian distribution $\mathcal{D}_y = \mathcal{N}(0, \sigma_y^2)$, there are many PUF instances for which y_i values are equal, and thus, they have the same architectural bias. Note that bias in PUF instance I_i would be large if $|y_i|$ is large.

Next, we evaluate delay PUF designs with respect to the fraction of PUF instances with small bias b_{archi}, which they can produce. To simplify the notation, we use b in the rest of this chapter to refer the architectural bias b_{archi} as we are not considering any other kind of biases in our discussion.

5 QUALITY EVALUATION OF DELAY PUFS USING ARCHITECTURAL BIAS

The delay PUF designs like APUF have been exploited in different security related applications, for example: *hardware-based authentication* and *secret-key generation.* An interesting property of PUF is its instance-specific random behavior. In the context of security applications, it is important to ensure that values of performance metrics (e.g., uniformity and reliability) for PUF instances of a PUF design are not significantly different from each other; otherwise different deployments of the same (PUF-enabled) application have different (nonuniform) security levels, which is not desirable. For example, let us consider an APUF-based key generation algorithm. If there are a few APUF instances with poor uniformity, then the entropy of their responses are also less, and it results in a weak key. One can think about post-silicon validation of the required metrics of a PUF instance before deployment, but in case of ASIC, it might result in a reduced yield rate when many PUF-enabled chips are rejected during validation. Thus, we need a pre-fabrication scheme that can estimate the fraction of good PUF instances (i.e. an instance that satisfies the required property of a metric) of a PUF design. Our objective in this section is to develop such a scheme based on the architectural bias of delay PUF designs.

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 ϵ, where $0 \leq \epsilon \leq 0.5$. This implies that acceptable value of b should satisfy $0 \leq b \leq \epsilon$ or equivalently by following Eq. (12), we have:

$$0.5 - \epsilon \leq B \leq 0.5 + \epsilon. \tag{14}$$

Based on Eq. (14), we now estimate a range for y that can ensure $0 \leq b \leq \epsilon$, as follows:

$$\begin{aligned} 0.5 - \epsilon \leq B \leq 0.5 + \epsilon \quad &\Rightarrow \Phi^{-1}(0.5 - \epsilon) \leq \frac{-y}{\sigma_Z} \leq \Phi^{-1}(0.5 + \epsilon) \\ \Rightarrow -\Phi^{-1}(0.5 + \epsilon) \leq \frac{y}{\sigma_Z} \leq -\Phi^{-1}(0.5 - \epsilon) &\Rightarrow -\sigma_Z\Phi^{-1}(0.5 + \epsilon) \leq y \leq -\sigma_Z\Phi^{-1}(0.5 - \epsilon). \end{aligned} \tag{15}$$

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

$$-T_y \leq y \leq T_y. \tag{16}$$

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

$$T_y = \sigma_Z\Phi^{-1}(0.5 + 0.1) = \frac{\sigma_Z}{4}. \tag{17}$$

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

Definition 5.1 (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 sampled from a Gaussian distribution lie in the range $[\mu - 3\sigma, \mu + 3\sigma]$. So, the most probable range of y is $[-3\sigma_y, +3\sigma_y]$, i.e.,

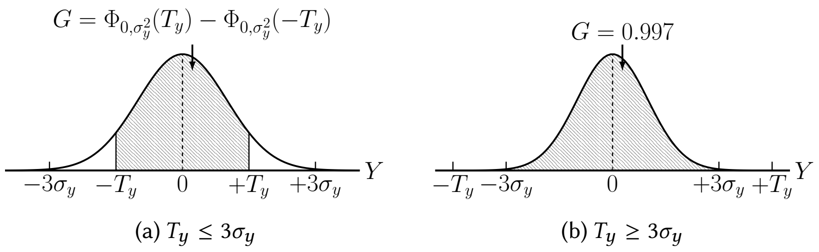


Fig. 6. Goodness of a PUF design w.r.t. fraction of good PUF instances, for two different cases. These two plots represent the distribution of bias values of a population of instances of a PUF design.

$\Pr(y \in [-3\sigma_y, +3\sigma_y]) \approx 0.997$. Let G be the fraction of good PUF instances, also called "Goodness Factor", and we compute it based on T_y as:

$$G = \begin{cases} \Phi_{0,\sigma_y^2}(T_y) - \Phi_{0,\sigma_y^2}(-T_y) = \Phi(\frac{\sigma_Z}{4\sigma_y}) - \Phi(-\frac{\sigma_Z}{4\sigma_y}), & \text{if} \quad T_y < 3\sigma_y \\ \Phi_{0,\sigma_y^2}(3\sigma_y) - \Phi_{0,\sigma_y^2}(-3\sigma_y) = \Phi(3) - \Phi(-3) \approx 0.997, & \text{if} \quad T_y \geq 3\sigma_y \end{cases} \tag{18}$$

This fact is also be depicted in Fig. 6. Although the generic definition of goodness factor is $G = \Phi_{0,\sigma_y^2}(T_y) - \Phi_{0,\sigma_y^2}(-T_y)$, we have considered two different cases as shown in Eq. (18) depending on the relation between T_y and σ_y because in case of $T_y \geq 3\sigma_y$, the value of G is significantly large, and thus, we can avoid the computation of expression $\Phi_{0,\sigma_y^2}(T_y) - \Phi_{0,\sigma_y^2}(-T_y)$. We use this fact later in Section 6.

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

Definition 5.2 (Good PUF Design vs. Bad PUF Design). For a given PUF design M_1, if the fraction of good PUF instances G_1 is greater than some threshold G_{th}, then PUF design M_1 is called as "**Good**" design. Otherwise, design M_1 is "**Bad**." For example, let $G_{\text{th}} = 0.90$. If the PUF design M_1 can generate more than 90% good PUF instances, then design M_1 is good. For a pair of PUF designs (M_1, M_2), if $G_1 > G_2$, then design M_1 is better than M_2.

Next, we discuss architectural bias in the context APUF, PAPUF and DAPUF designs, and compare these designs based on their architectural bias values. For these designs, architectural bias will be estimated based on their analytic delay models.

6 ARCHITECTURAL BIAS IN APUF VARIANTS

As case studies of theory developed in Section 4, we consider APUF variants to compare their architectures. To explain the architectural bias (cf. Eq. (13)), we consider linear additive delay models of APUF, PAPUF and DAPUF (cf. Section 2.2). A random variable $Z \sim \mathcal{D}_Z = \mathcal{N}(\mu_Z, \sigma_Z^2)$ would be used to represent $\Delta(n-1)$. In general, Z can be expressed using delay model as:

$$Z = \left(\sum_{i=0}^{k-2} \mathbf{w}[i]\mathbf{\Phi}[i]\right) + \mathbf{w}[k-1] = X + y, \tag{19}$$

where Gaussian random variable $X = \sum_{i=0}^{k-2} \mathbf{w}[i]\mathbf{\Phi}[i]$, $y = \mathbf{w}[k-1]$, and k is the size of weight vector $\mathbf{w}$ used to define $\Delta(n-1)$ in Eqs. (1), (5) and (7). Each $\mathbf{w}[i], i \in [0 : k-1]$ is sampled from a Gaussian distribution $\mathcal{D}_{w_i} = \mathcal{N}(0, \sigma_{w_i}^2)$, (i.e. $\mathbf{w}[i] \leftarrow \mathcal{D}_{w_i}$), assuming that delay parameters p_i, q_i, r_i, s_i of the i-th APUF's switch (cf. Fig. 2) are sampled independently from a Gaussian distribution with mean $\mu = 0$ and variance σ^2, i.e., $p_i, q_i, r_i, s_i \leftarrow \mathcal{N}(0, \sigma^2)$ [5]. The value of $\sigma^2_{w_i}$ depends on PUF architecture and process variation, as definition of $\mathbf{w}[i]$ is architecture specific. Since the value of X for a challenge is sum of $\mathbf{w}[0], \dots, \mathbf{w}[k-2]$, and feature vector $\mathbf{\Phi}$ only decides the polarities of $\mathbf{w}[i]$'s, $i \in [0 : k-2]$, X would follow a Gaussian distribution $\mathcal{D}_X = \mathcal{N}(0, \sigma^2_X)$, where $\sigma^2_X = \sigma^2_{w_0} + \cdots + \sigma^2_{w_{k-2}}$. For each PUF instance, y value is sampled independently from the Gaussian distribution $\mathcal{D}_y = \mathcal{D}_{w_{k-1}}$. Thus, $Z \sim \mathcal{D}_Z = \mathcal{N}(y, \sigma^2_Z)$, where $\sigma^2_Z = \sigma^2_X$. For all instances of a PUF design, σ^2_Z would be the same, but y values are sampled from $\mathcal{D}_y = \mathcal{N}(0, \sigma^2_y)$.

Next, we estimate the parameters of σ^2_X and σ^2_y for APUF variants based on their delay models.

6.1 Architectural Bias and Goodness Factor of APUF

Let $Z_A = \Delta(n-1)$ be a random variable representing the delay differences of an APUF instance and it follows a Gaussian distribution $\mathcal{N}(\mu_A, \sigma^2_A)$. Based on the model of APUF in Eq. (1), we can write:

$$Z_A = \left(\sum_{i=0}^{n-1} \mathbf{w}[i]\mathbf{\Phi}[i] \right) + \mathbf{w}[n] = X_A + y_A,$$

where $X_A = \sum_{i=0}^{n-1} \mathbf{w}[i]\mathbf{\Phi}[i]$ is challenge dependent and $y_A = \mathbf{w}[n]$ is challenge independent. From Eq. (3), we have:

(1) $\mathbf{w}[0] \leftarrow \mathcal{D}_{w_0} = \mathcal{N}(0, 2\sigma^2)$, $\mathbf{w}[n] \leftarrow \mathcal{D}_{w_n} = \mathcal{N}(0, 2\sigma^2)$, and $\mathbf{w}[i] \leftarrow \mathcal{D}_{w_i} = \mathcal{N}(0, 4\sigma^2)$, for $i = 1, \dots, n-1$,
(2) $X_A \sim \mathcal{N}(0, \sigma^2_A)$ where $\sigma^2_A = (4n-2)\sigma^2 \approx 4n\sigma^2$ for moderately large values of n,
(3) $y_A \leftarrow \mathcal{D}_{y_A} = \mathcal{D}_{w_n} = \mathcal{N}(0, \sigma^2_{y_A})$, where $\sigma^2_{y_A} = 2\sigma^2$.

Thus, $Z_A = X_A + y_A \sim \mathcal{N}(y_A, \sigma^2_A)$. Based on Eq. (15), we can compute the threshold for acceptable y_A values as:

$$T_{y_A} = \sigma_A/4 = \sqrt{n}\sigma/2.$$

In case of Gaussian distribution $\mathcal{D}_{y_A}$, we have $3\sigma_{y_A} = 3\sqrt{2}\sigma$. Since $T_{y_A} > 3\sigma_{y_A}$ for $n > 72$, the fraction of good APUF instances (cf. Eq. (18)) is:

$$G_A \approx \mathbf{0.997}.$$

Later, we show using simulation results in Section 9 (cf. Table 2 and Fig. 9) that the distribution of bias for APUF has a mean close to 0, and the variance is significantly small. Hence, we would say that the APUF is relatively free from architectural bias.

6.2 Architectural Bias and Goodness Factor of PAPUF

Let $Z_P = \Delta(n-1)$ be a Gaussian random variable representing the delay differences of a PAPUF instance. From Eq. (5), we can write:

$$Z_P = \left(\sum_{i=0}^{n-1} \mathbf{w}[i]\mathbf{\Phi}[i] \right) + \mathbf{w}[n] = X_P + y_P.$$

Similar to APUF, $X_P = \sum_{i=0}^{n-1} \mathbf{w}[i]\mathbf{\Phi}[i]$ is challenge dependent, and $y_P = \mathbf{w}[n]$ is challenge independent. We also have following information from Eq. (5):

(1) $\mathbf{w}[i] \leftarrow \mathcal{D}_{w_i} = \mathcal{N}(0, 2\sigma^2)$ for $i = 0, \dots, n-1$, and $\mathbf{w}[n] \leftarrow \mathcal{D}_{w_n} = \mathcal{N}(0, 2n\sigma^2)$,
(2) $X_P \sim \mathcal{N}(0, \sigma^2_P)$ where $\sigma^2_P = 2n\sigma^2$, and
(3) $y_P \leftarrow \mathcal{D}_{y_P} = \mathcal{D}_{w_n} = \mathcal{N}(0, \sigma^2_{y_P})$ where $\sigma^2_{y_P} = 2n\sigma^2$.

Thus, $Z_P = X_P + y_P \sim \mathcal{N}(y_P, \sigma^2_P)$. Based on Eq. (15), we can compute the threshold for acceptable y_P values as:

$$T_{y_P} = \sigma_P/4 = \sqrt{2n}\sigma/4.$$

In case of Gaussian distribution $\mathcal{D}_{y_{\rm P}}$, we have $3\sigma_{y_{\rm P}} = 3\sqrt{2n}\sigma$. Thus, $T_{y_{\rm P}} < 3\sigma_{y_{\rm P}}$, and the fraction of good PAPUF instances (cf. Eq. (18)) is:

$$G_{\rm P} = \Phi(\frac{\sigma_{\rm P}}{4\sigma_{y_{\rm P}}}) - \Phi(-\frac{\sigma_{\rm P}}{4\sigma_{y_{\rm P}}}) = \Phi(\frac{\sqrt{2n}\sigma}{4\sqrt{2n}\sigma}) - \Phi(-\frac{\sqrt{2n}\sigma}{4\sqrt{2n}\sigma}) = \Phi(1/4) - \Phi(-1/4) \approx \mathbf{0.20}.$$

Thus, it is expected that the distribution of bias for PAPUF would have significantly large standard deviation (cf. Fig. 9c). 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 [3, 12].

6.3 Architectural Bias and Goodness Factor of DAPUF

Let $Z_{\rm D} = \Delta(n-1)$ be a Gaussian random variable representing the delay differences of a DAPUF instance. From Eq. (7), we can write

$$Z_{\rm D} = \left(\sum_{i=0}^{2n-2} \mathbf{w}[i]\mathbf{\Phi}[i]\right) + \mathbf{w}[2n-1] = X_{\rm D} + y_{\rm D},$$

where $X_{\rm D} = \sum_{i=0}^{2n-2} \mathbf{w}[i]\mathbf{\Phi}[i]$ and $y_{\rm D} = \mathbf{w}[2n-1]$. From the definition of $\mathbf{w}[i]$ in Eq. (7), we have:

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

Thus, $Z_{\rm D} = X_{\rm D} + y_{\rm D} \sim \mathcal{N}(y_{\rm D}, \sigma_{\rm D}^2)$. Based on Eq. (15), 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 distribution $\mathcal{D}_{y_{\rm D}}$, we have $3\sigma_{y_{\rm D}} = 3\sqrt{2n}\sigma$. Thus, $T_{y_{\rm D}} < 3\sigma_{y_{\rm D}}$, and the fraction of good DAPUF instances (cf. Eq. (18)) is:

$$G_{\rm D} = \Phi(\frac{\sigma_{\rm D}}{4\sigma_{y_{\rm D}}}) - \Phi(-\frac{\sigma_{\rm D}}{4\sigma_{y_{\rm D}}}) = \Phi(\frac{\sqrt{6n}\sigma}{4\sqrt{2n}\sigma}) - \Phi(-\frac{\sqrt{6n}\sigma}{4\sqrt{2n}\sigma}) = \Phi(\sqrt{3}/4) - \Phi(-\sqrt{3}/4) \approx \mathbf{0.335}.$$

Hence, from the above analysis we conclude that the standard deviation of bias distribution of DAPUF is also large, but less than that of PAPUF. In case of DAPUF design, there are 13.5% more "good" instances compared to PAPUF.

6.4 Computation of Goodness Factor without Delay Model

Although our discussion on the estimation of goodness factor is based on the delay models for APUF variants, this approach would not be feasible for the delay PUF for which analytical delay model is not available. However, estimation of G can also be done based on the CRP dataset of a PUF design, and we have explained the computation steps in Algorithm 1. Algorithm 1 takes the followings as inputs: (a) ϵ as bias tolerance value, (b) K is number of instances of PUF design P, (c) N is number of challenges. The value of K should be large enough to ensure the correct estimation of G. To observe the architectural bias in a delay PUF design, one can apply Algorithm 1 on the CRPs of simulated PUF instances; physical implementation is not necessary. If the value of G is significantly large for simulation of the PUF design, then one can proceed for actual silicon implementation which is expensive.

7 IMPACTS OF ARCHITECTURAL BIAS ON UNIQUENESS AND RELIABILITY METRICS

According to Eq. (12), uniformity is inversely proportional to bias, i.e., a PUF instance with less architectural bias implies that PUF instance has good uniformity metric. So, in this section, we discuss only the effects of architectural bias on uniqueness and reliability metrics.

7.1 Uniqueness Metric

The *uniqueness* metric [10], also known as inter HD, is used as an estimate of average dissimilarity in challenge-response behavior of PUF instances with respect to the same set of challenges. It can be defined as an average pairwise Hamming distance between the responses of a population of PUF instances. In case of a PUF design where uniformity of PUF instances are significantly less, distribution of pairwise HD values exhibits a large standard deviation. Ideally, distribution of pairwise HD should have the mean value close to 50% with small standard deviation, as this allows to identify maximum number of unique chips based on PUF response. In general, a *PUF design with large architectural bias results in poor uniqueness*, and this fact can also be observed from simulation result in Section 9. There might be some special cases, where uniformity values two PUF instances are 50%, but their HD distance is 100%. For example, let $\mathbf{r}_1$ = "1010101010" and $\mathbf{r}_2$ = "0101010101" be 10-bit responses of two PUF instances obtained from 10 different challenges, and the same set of challenges are applied to both the instances. In both the cases, they have 50% uniformity, but uniqueness is 100% as $\mathbf{r}_1$ and $\mathbf{r}_2$ differ at each bit position. However, these cases are rare.

This fact implies that uniqueness of an ideal APUF is significantly better than PAPUF and DAPUF, and DAPUF has comparatively better uniqueness than that of PAPUF.

7.2 Reliability Metric

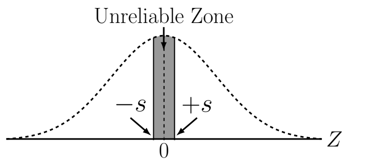
The reliability of a PUF is the ability to generate the same response to a given challenge repeatedly over different time instants and operating conditions, e.g. variations in ambient temperature and supply voltage. For a given challenge $\mathbf{c}$, if delay difference is significantly small, then polarity of delay difference can change with high probability due to noise.

In practical use of PUF, 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 $z_\mathbf{c}$ due to a given challenge $\mathbf{c}$ lies in the interval $z_\mathbf{c} \in [-s, +s]$, then response to challenge $\mathbf{c}$ is treated as

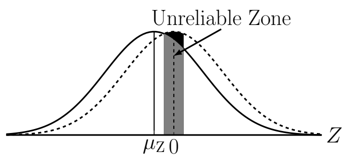
Algorithm 1 ϵ-Bound Goodness Factor Computation of a Delay PUF Design

Input: P—A PUF design; ϵ—bias tolerance value; K—number of instances of P; N—number of challenges
Output: Goodness factor G of PUF design P

```
1:  $C$ := Set of $N$ randomly generated challenges of PUF $P$
2:  $\mathbf{b}$ := An $K$-component bias vector initialized with zeros
3:  $Gcnt = 0$ ▷ No. of good PUF instances
4:  for $i = 1$ to $K$ do
5:      Create the $i$-th instance $P_i$ of PUF design P
6:      $r1Cnt := 0$ ▷ Count no. of 1's
7:      for all $\mathbf{c} \in C$ do
8:          $r := P_i(\mathbf{c})$ ▷ Evaluate $P_i$ with challenge $\mathbf{c}$
9:          if $r = 1$ then
10:             $r1Cnt := r1Cnt + 1$
11:     $\mathbf{b}[i] = |0.5 - \frac{r1Cnt}{N}|$ ▷ $|x|$– absolute value of $x$
12: for $i = 1$ to $K$ do
13:     if $\mathbf{b}[i] \le \epsilon$ then
14:         $Gcnt := Gcnt + 1$ ▷ Good PUF instance
15: $G = \frac{Gcnt}{K}$ ▷ Goodness factor
```



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



(b) Reduced unreliable zone (gray region) under the curve of Z (curve with solid line) due to shift in μ_Z from its ideal value 0.

Fig. 7. Demonstration of reliability of a delay PUF instance. (a) If delay difference $z_{\mathbf{c}}$ of challenge $\mathbf{c}$ lies in $[-s, +s]$, then response to challenge $\mathbf{c}$ is unreliable. (b) Showing the shift in the mean of delay difference distribution Z due to architectural issue, and it results in reduction of unreliable area under the curve of Z. The amount of unreliability reduction occurred is equivalent to the area of black region.

unreliable. In [1, 11], authors discussed about 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 $\mu_Z = 0$, then the unreliable zone under the curve of Z is reduced. So, if $\mu_Z \approx 0$ for a PUF instance, then its reliability property is poor compared to the PUF instance having $|\mu_Z| > 0$.

Since the architectural bias values are smaller for most of the APUF instances compared to PAPUF and DAPUF, it exhibits poor reliability than PAPUF and DAPUF. The PAPUF has better reliability compared to DAPUF. Since architectural bias value is instance-specific, the reliability metric would be different for PUF instances of a PUF design. Thus, we need to consider both the mean and standard deviation of reliability values of a population of instances to evaluate the reliability metric of a PUF design.

Next, we discuss the security properties of APUF, PAPUF and DAPUF. Note that, in security analysis, we exploit the linear delay models of APUF variants, which depend on their architectures. In this paper, we could not establish the relation between the architectural bias and security properties. We consider this as our future work on this topic.

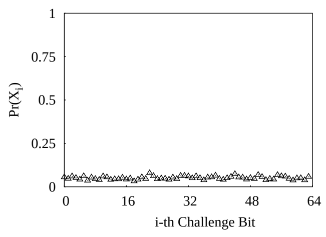
8 SECURITY ANALYSIS

In this section, we discuss the security features of these architectural variants of APUF. Our security analysis would focus on the differential cryptanalysis based on the SAC property [14, 17], and machine learning based modeling attack [19]. In the literature, one can find security analysis of APUF using these two approaches. However, we present the security properties of PAPUF and DAPUF for the sake of comparison with APUF.

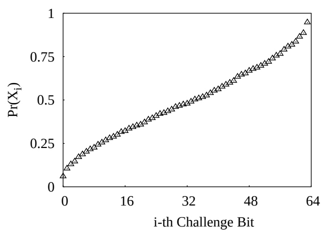
8.1 Differential Cryptanalysis Based on SAC Property

An n-bit input, 1-bit output Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is said to satisfy SAC property, if the value of the function f complements with probability of one-half, whenever a single bit of the input is complemented [18]. For the sake of explanation, we consider a random variable X_i defined as:

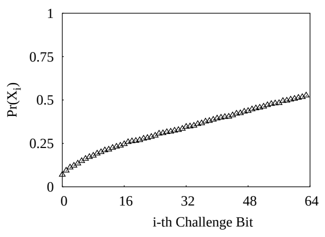
$$X_i = \begin{cases} 1, & \text{If output of } f \text{ complemets due to complemented } i\text{-th challenge bit} \\ 0, & \text{otherwise.} \end{cases}$$



(a) 64-bit PAPUF



(b) 64-bit APUF



(c) 64-bit DAPUF

Fig. 8. Depicting SAC properties of 64-bit simulated APUF, PAPUF and DAPUF.

The probability $\Pr(X_i = 1)$ is the probability that output f complements when i-th bit of challenge is complemented. To satisfy the SAC property, a PUF design with n-bit input should achieve $\Pr(X_i = 1) = 0.5$ for $i = 0, \ldots, n-1$.

Figure 8 shows $\Pr(X_i = 1)$ for APUF design variants with 64-bit challenge, and these designs are simulated in Matlab (cf. Section 9). While the $\Pr(X_i = 1)$ value lies in [0,1] for APUF, it lies in [0,0.5] in cases of both the PAPUF and DAPUF. Note that small as well as the large values for $\Pr(X_i = 1)$ are not desirable from security perspective as an adversary can generate many related CRPs based on $\Pr(X_i = 1)$ value [17]. For example, let us consider a CRP $(\mathbf{c}, r)$ and a challenge $\mathbf{c}_i$, where $\mathbf{c}$ and $\mathbf{c}_i$ differ only at i-th bit position. Let us denote r_i as the response to challenge $\mathbf{c}_i$ and the adversary can predict the value of r_i based on $(\mathbf{c}, r)$ as follows:

$$r_i = \begin{cases} r, & \Pr(X_i = 1) \longrightarrow 0 \\ r \oplus 1, & \Pr(X_i = 1) \longrightarrow 1. \end{cases}$$

In case of PAPUF, $\Pr(X_i = 1)$ value is significantly small and $\Pr(X_i = 1)$ value is a similar for all challenge bit positions. Thus, the number of related CRPs can be generated is significantly larger for PAPUF. On the other hand, in case of DAPUF, $\Pr(X_i = 1)$ value is small for least significant bit (LSB) positions and it approaches 0.5 for most significant bit (MSB) positions of challenge. As a consequence of SAC property, PAPUF leaks more information to adversary when partial CRP set of its is revealed, regardless of the complemented challenge bit positions. In case of APUF, both the LSB and MSB positions are are useful to the adversary to generate related CRPs, whereas only LSB positions are useful in the case of DAPUF. Thus, DAPUF has comparatively better SAC property than that of APUF and PAPUF, and the SAC property of PAPUF is significantly poor compared to both the APUF and DAPUF.

8.2 Machine Learning Based Modeling

Like APUF, both the PAPUF and DAPUF have a linear additive analytic model (cf. Section 2.2), and thus, these are also vulnerable to machine learning based modeling attack. An important step in machine learning is to decide what would be the feature set (or independent inputs) of the machine learning algorithm. From (cf. Section 2.2), it can be observed that modeling of these APUF designs need a different feature set, summarized as follows:

- PAPUF modeling uses the challenge of PUF as feature set with an additional constant 1 bit. Thus, no transformation is required to get the feature set from challenge. For n-bit challenge, the size of feature set is $n + 1$.
- APUF modeling uses the parity vector of challenge as feature set. Like PAPUF, the size of feature set is $n + 1$ for an n-bit challenge.

Table 1. Modeling accuracy values of LR-based models for 64-bit simulated APUF variants

PUF	# CRPs	Accuracy (%)			
		max.	avg.	std.	NMGE_90[†]
PAPUF	5000	96.62	77.14	22.17	6/10
	10000	98.58	83.59	21.99	7/10
	15000	98.98	83.93	22.22	7/10
APUF	5000	99.55	99.40	0.06	10/10
	10000	99.83	99.74	0.04	10/10
	15000	99.86	99.82	0.03	10/10
DAPUF	5000	99.09	98.80	0.17	10/10
	10000	99.57	99.45	0.06	10/10
	15000	99.67	99.58	0.05	10/10

[†] **NMGE_90** is denoted as a ratio a/b, where a is the number of runs that results in a model with accuracy $\geq$ 90%, and b denotes the total number of runs of LR algorithm considered during the model building.
Note : In testing phase, we have used 30×10^3 CRPs.

- DAPUF's feature set consists of both the challenge bits and its parity bits. In this case, the size of feature set is $2n$-bit which is derived from n-bit challenge.

Table 1 reports the modeling accuracy values of Logistic Regression (LR) [19] based models of APUF design variants. The reported modeling results are based on the CRPs of a randomly generated simulated instance of 64-bit APUF design variants, and PUF instances are 100% reliable. For detailed simulation setup, we refer the reader to Section 9. We have considered 10 different runs of LR algorithm with different CRP set of a given size. It can be observed from Table 1 that for PAPUF, each run of LR algorithm could not produce a good model, where a good model is a LR-based model with modeling accuracy at least 90%. On the other hand for DAPUF and APUF, each run of LR algorithm generated a good model, provided that CRP sets are selected randomly.

Although we have obtained models with accuracy greater than 99.50% for APUF and DAPUF with 15,000 CPRs, the accuracy of PAPUF model is less than 99% even though the same size of training CRP set is used. One reason is that random CRPs used in modeling of PAPUF might be correlated/similar due to large architectural bias (i.e. poor uniformity). Thus, modeling accuracy heavily depends on the number of useful CRPs in the training CRP set, instead of the size of training CRP set. As reported in Table 1, for PAPUF, only 6-7 models achieves modeling accuracy greater than 90% out of 10 models obtained from 10 different runs of LR algorithm.

Both the APUF and PAPUF require a similar computation time for building a model, whereas DAPUF requires more computation time as it has more parameters compared to APUF and PAPUF. However, the modeling time of DAPUF is linearly related to the modeling time of APUF. Results in Table 1 might give a false impression to the reader that PAPUF has slightly better modeling resistance than both the APUF and DAPUF, but this is not true. The reason behind the less modeling accuracy for PAPUF model compared to APUF and DAPUF models is the existence of many related CRPs in the training set of PAPUF. We need to consider more CRPs in the training for PAPUF than that of APUF and DAPUF to achieve the same modeling accuracy.

9 SIMULATION RESULTS

To validate the notion of architectural bias and its effects on performance metrics of APUF, PAPUF and DAPUF, in this section we present simulation based results.

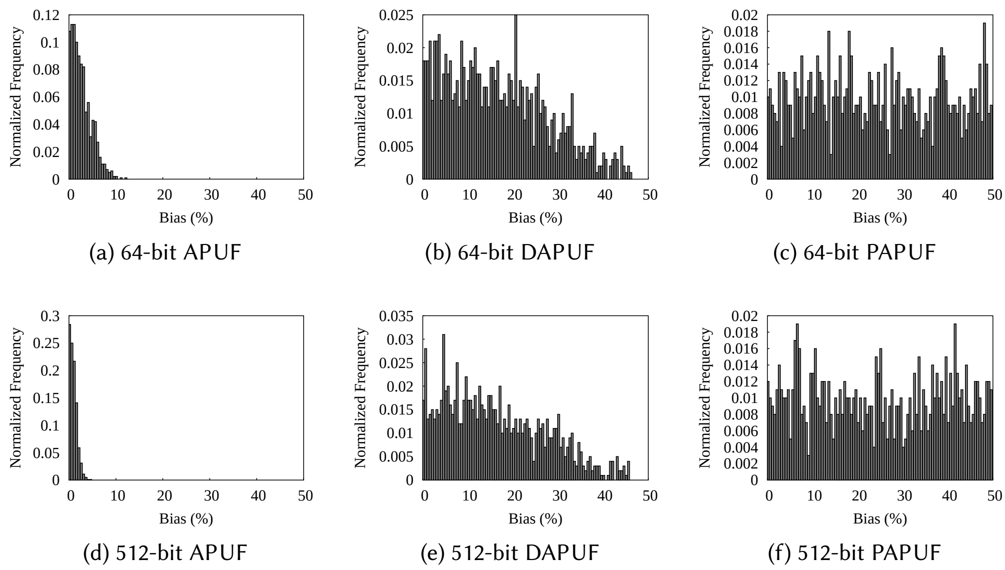


Fig. 9. Histograms of bias values of simulated APUF, PAPUF and DAPUF based on the bias values of 1000 instances of each PUF design. In this case, all instances are perfectly reliable, i.e., $\sigma_{\text{noise}} = 0$.

Simulation Setup. We simulated the APUF variants using Matlab (for challenge sizes of 64-bit, 128-bit, 256-bit and 512-bit), assuming that propagation delay of each delay component follows normal distribution with $\mu = 10$ and $\sigma = 0.05$. In case of simulation, we directly compare the delays of 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. *Note that in case of PAPUF simulation, tuning elements are not considered.*

To observe the effect of architectural bias on the reliability metric of simulated PUF, we have considered a simulated additive noise (in case of hardware implementation this noise occurs due to temperature and supply voltage variations) with the delay distribution of delay components. In the presence of noise, each delay component follows a normal distribution $\mathcal{N}(\mu + 0, \sigma^2 + \sigma^2_{\text{noise}})$, where noise distribution is a normal distribution $\mathcal{N}(0, \sigma^2_{\text{noise}})$. In our simulations, we have used following relation between σ and σ_{noise} to control the reliability levels: $\sigma_{\text{noise}} = \gamma\sigma$, where $0 \leq \gamma \leq 1$. For $\gamma = 0$, PUF instance is 100% reliable.

We simulated 1000 instances for each PUF design, and uniformity, uniqueness and reliability metrics were computed based on these population of PUF instances using 50,000 CRPs. In the presence of noise, golden responses of a simulated PUF instance are generated using majority voting of responses to the same set of challenge over 25 measurements. Reliability of a simulated PUF instance, with a given value of $\sigma_{\text{noise}} = p$, is estimated as the Hamming distance between the golden responses at $\sigma_{\text{noise}} = p$ and reference responses at $\sigma_{\text{noise}} = 0$.

Distribution of Bias. Figure 9 depicts the distribution of bias values of 1000 instances for APUF, DAPUF and PAPUF designs, and bias of a PUF instance is computed as the difference (only magnitude of difference is considered) of numbers of 0's and 1's in the PUF responses. As discussed earlier, PUF instances with bias values closer to zero are good. It can be observed from Fig. 9 that range of bias values for APUF instances are significantly small compared to the DAPUF and PAPUF, and a smaller range started with ‘0’ is desirable. From Figs. 9a to 9c, it can be observed that bias distributions of APUF and DAPUF follow (half) normal distributions with mean close to zero, and this property along with a small standard deviation of bias distribution is a desirable property for any good PUF design. However, in case of PAPUF, the mean of bias distribution is significantly greater than zero and standard deviation is large. Thus, the range of bias values is significantly larger for PAPUF than that of APUF and DAPUF.

From Fig. 9, it can be observed that there is a dependency of bias distribution on the challenge length. By comparing Figs. 9a and 9d, it is observed that standard deviation of bias distribution for 512-bit APUF is less compared to 64-bit APUF, while a (partially) reverse trend can be observed for PAPUF and DAPUF. In this context, APUF with larger challenge is preferable.

In Table 2, we have reported the number of good PUF instances (G) in the populations of 1000 simulated PUF instances for different settings of bias tolerance parameter ϵ. From this result, it can be observed that, for $\epsilon = 0.1$ (i.e. at most 10% bias is acceptable), average values of G (i.e., fraction of PUF instances with bias values at most ϵ) over different challenge sizes for APUF, DAPUF and PAPUF are 100%, 33.75% and 20.25%, respectively, and it matches the theory developed in Section 6. Note that the values of G are computed using CRPs of simulated PUF instances by following Algorithm 1.

Uniformity Results. Uniformity is the probability that a PUF instance produces ‘1’ as response. In case of a population of PUF instances of a PUF design, the distribution of uniformity values should follow a normal distribution with mean close to 50% (i.e, uniform distributions of 0’s and 1’s) with small standard deviation. Uniformity distributions of APUF, PAPUF and DAPUF are shown in Fig. 10.

From Figs. 10a and 10b, it is evident that uniformity distributions of 64-bit APUF and DAPUF have mean close to 50%. Form the uniformity distribution plot of 64-bit PAPUF in Fig. 10c, it is hard to observe the mean value, but it has also mean close to 50% as shown in Table 3. Thus, if one considers only mean (avg. in Table 3), then all these APUF variants have good uniformity metrics, but that is not true. Uniformity distribution of PAPUF in Fig. 10c shows that there are many PAPUF instances with uniformity values approximately 0 and 1. Although, for PAPUF, uniformity distribution has the mean value close to 50%, standard deviation is significantly large. The standard deviation of PAPUF’s uniformity distribution is larger than that of DAPUF, and DAPUF’s uniformity distribution has larger standard deviation than that of APUF (cf. Table 3). This implies that majority of APUF instances have good uniformity values, whereas for PAPUF, majority of instances have poor uniformity values.

In addition, it can be observed from Table 3, standard deviation of uniformity distribution reduces with increasing challenge size for APUF, but that trend is not prominent for DAPUF and PAPUF design. This implies that an APUF with longer challenge has a better uniformity metric compared

Table 2. Percentage of “good” PUF instances (G) in the populations of 1000 simulated instances

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	18	33	49	63
	PAPUF	10	20	31	42
512	APUF	100	100	100	100
	DAPUF	18	35	51	65
	PAPUF	11	22	32	42

Note The ϵ is the bias tolerance parameter as defined in Section 5.

Table 3. Uniformity and Uniqueness metrics (%) of simulated APUF, PAPUF and DAPUF

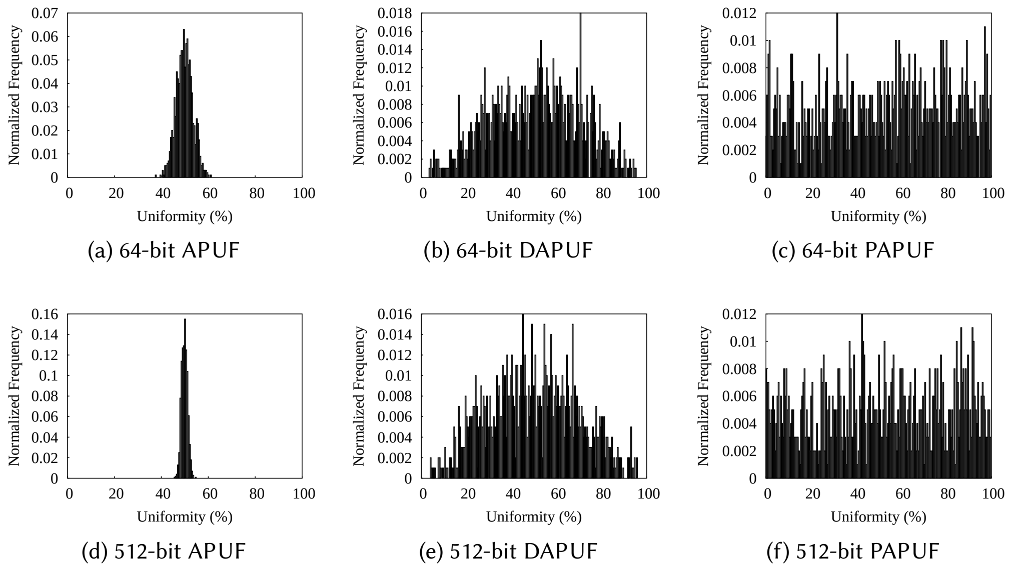
$n^{\dagger}$	**Uniformity**			**Uniqueness**		
	APUF	DAPUF	PAPUF	APUF	DAPUF	PAPUF
	(Avg., Std.)	(Avg., Std.)	(Avg., Std.)	(Avg., Std.)	(Avg., Std.)	(Avg., Std.)
64	(50.04, 3.53)	(50.59, 19.96)	(51.98, 28.65)	(49.96, 4.01)	(50.10, 8.29)	(49.90, 16.51)
128	(49.78, 2.52)	(50.30, 20.38)	(50.80, 29.15)	(49.95, 2.79)	(50.03, 8.44)	(50.11, 17.09)
256	(50.07, 1.73)	(49.80, 19.78)	(49.89, 28.59)	(50.03, 2.01)	(50.03, 7.99)	(50.14, 16.31)
512	(50.07, 1.73)	(49.61, 19.66)	(51.14, 28.91)	(50.03, 2.01)	(49.85, 8.00)	(50.08, 16.70)

† Challenge size in bits

to an APUF with small challenge.

Uniqueness Result. Uniqueness is computed as the average pairwise HD of responses over a population of instances of a PUF design. Here, we see that this average value does not provide the clear picture about the distribution of pairwise HD values. Ideally, the distribution of pairwise HD distribution of a good PUF design should follow a Gaussian distribution with mean close to 50% with sufficiently small standard deviation. Uniqueness distributions of APUF, PAPUF and DAPUF are reported in Fig. 11 and Table 3. Like uniformity metric, mean values of uniqueness distributions are close to 50% for these APUF design variants. However, standard deviation of uniqueness distribution for PAPUF is large compared to DAPUF. In case of APUF, this standard deviation is significantly small, and it reduces with increasing challenge length (cf. Table 3).

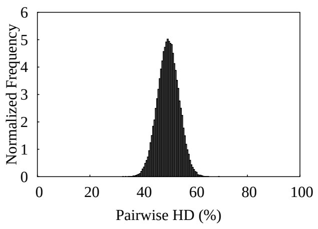
Reliability Results. According to Section 7.2, large bias implies a better reliability. Ideally, reliability distribution of a PUF design should follow a Gaussian distribution with mean at 100% (i.e.,



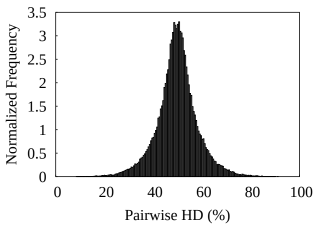
(a) 64-bit APUF (b) 64-bit DAPUF (c) 64-bit PAPUF

(d) 512-bit APUF (e) 512-bit DAPUF (f) 512-bit PAPUF

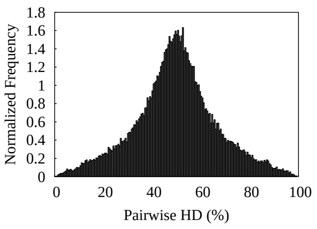
Fig. 10. Histograms of uniformity values of simulated APUF, PAPUF and DAPUF based on the uniformity values of 1000 instances of each PUF design. In this case, all instances are perfectly reliable, i.e., $\sigma_{\text{noise}} = 0$.



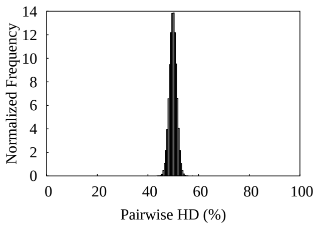
(a) 64-bit APUF



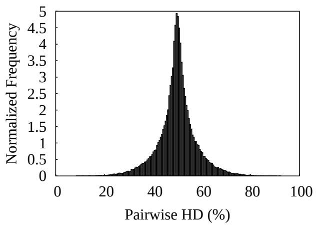
(b) 64-bit DAPUF



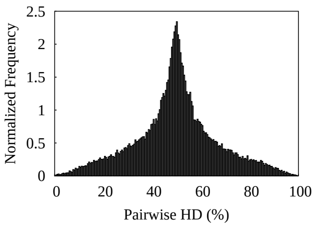
(c) 64-bit PAPUF



(d) 512-bit APUF

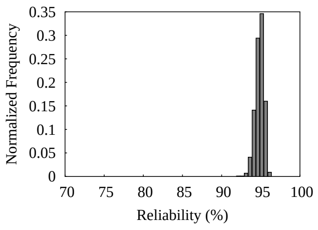


(e) 512-bit DAPUF

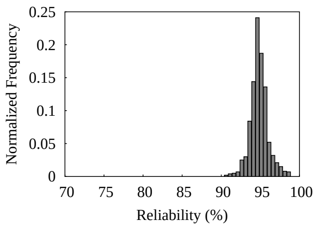


(f) 512-bit PAPUF

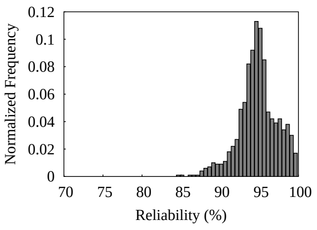
Fig. 11. Histograms of uniqueness values of simulated APUF, PAPUF and DAPUF based on the responses of 1000 instances of each PUF design. Since the value $\binom{1000}{2}$ is a large, we considered only 1,00,000 pairs in each histogram, instead of all possible pairs. In this case, all instances are perfectly reliable, i.e., $\sigma_{\text{noise}} = 0$.



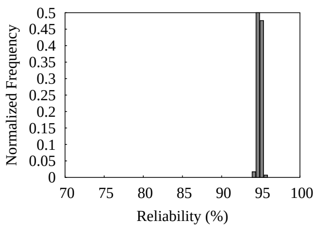
(a) 64-bit APUF



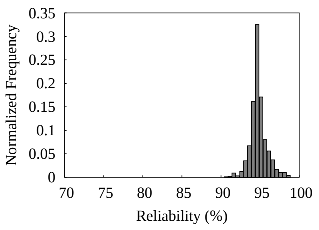
(b) 64-bit DAPUF



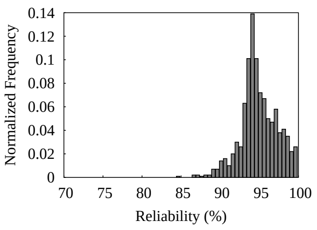
(c) 64-bit PAPUF



(d) 512-bit APUF



(e) 512-bit DAPUF



(f) 512-bit PAPUF

Fig. 12. Histograms of reliability values of simulated APUF, PAPUF and DAPUF based on the reliability values of 1000 instances of each PUF design. In this case, we considered $\sigma_{\text{noise}} = \sigma/2$.

Table 4. Reliability metrics (%) of simulated APUF, PAPUF and DAPUF designs

$n^{\dagger}$	$\gamma^{\ddagger}$	**APUF (Avg., Std.)**	**DAPUF (Avg., Std.)**	**PAPUF (Avg., Std.)**
64	1/2	(94.97, 0.54)	(94.95, 1.14)	(94.91, 2.45)
	1/20	(99.49, 0.06)	(99.49, 0.12)	(99.49, 0.25)
	1/80	(99.87, 0.02)	(99.87, 0.03)	(99.87, 0.06)
128	1/2	(94.97, 0.39)	(95.00, 1.14)	(94.99, 2.39)
	1/20	(99.49, 0.05)	(99.49, 0.12)	(99.49, 0.25)
	1/80	(99.87, 0.02)	(99.87, 0.03)	(99.87, 0.06)
256	1/2	(94.97, 0.29)	(94.94, 1.11)	(94.86, 2.43)
	1/20	(99.49, 0.04)	(99.49, 0.11)	(99.49, 0.25)
	1/80	(99.87, 0.02)	(99.87, 0.03)	(99.87, 0.06)
512	1/2	(94.98, 0.21)	(94.92, 1.04)	(95.03, 2.35)
	1/20	(99.49, 0.04)	(99.49, 0.12)	(99.50, 0.24)
	1/80	(99.87, 0.02)	(99.87, 0.03)	(99.87, 0.06)

† Challenge size in bits
‡ $\sigma_{\text{noise}} = \gamma\sigma$, where σ is the standard deviation of delay distribution of delay component.

if one considers probability value, then mean should be at 1.0) with sufficiently small standard deviation. Figure 12 shows the reliability distributions of APUF, PAPUF and DAPUF for $\sigma_{\text{noise}} = \sigma/2$ (i.e., $\gamma = 1/2$), and reliability results for different noise levels and challenge length are provided in Table 4. Figures 12c and 12f show the reliability distributions of 64-bit and 512-bit PAPUFs, respectively. It can be observed that there are many PAPUF instances with very good reliability values, while this quantity is significantly less for DAPUF and APUF. In case of PAPUF, number of reliable instances increases with the increasing challenge length. Although the standard deviation of APUF's reliability distribution (cf. Figs. 12a and 12d) is significantly small, but its mean value is significantly shifted away from the ideal mean close to 100%. In addition, number of APUF instances with comparatively good reliability values reduces with the increasing challenge length. Reliability distribution of DAPUF is comparatively better than that of APUF.

This implies that zero bias is not good in practice as it results in poor reliability. Instead, bias values should be small such that there is a good trade-off among all performance metrics, namely uniformity, uniqueness and reliability.

10 COMPARISON OF APUF VARIANTS

Before comparing the PAPUF with APUF and DAPUF, we like to mention that we have considered PAPUF without tuning bits to maintain architectural similarity with APUF and DAPUF. We discuss the effects of tuning bits later in this section. From Table 2, it is evident that architectural bias in APUF is comparatively less than that of DAPUF and PAPUF, and as a consequence of this, most of the APUF instances have very good uniformity. On the other hand, PAPUF and DAPUF have large architectural bias, and the architectural bias in DAPUF is less compared to PAPUF.

From the perspective of uniformity and uniqueness metrics, APUF is superior than PAPUF and DAPUF, and DAPUF is better than PAPUF. Since less bias implies poor reliability, reliability of APUF is comparatively lesser than PAPUF and DAPUF, and reliability of DAPUF is less compared to PAPUF.

From the traditional ML-based modeling attack perspective, APUF, PAPUF and DAPUF are vulnerable to modeling attack with asymptotically similar data and time complexities, as they have additive linear delay models. The SAC property of PAPUF is significantly poor as output transition probability is a similar regardless of the challenge bit positions. On the other hand, output transition probability in DAPUF and APUF depend on the challenge bit positions. Among these APUF design variants, DAPUF exhibits better SAC property than that of APUF and PAPUF. An implementation-platform independent comparison of performance and security metrics for APUF, PAPUF and DAPUF are summarized in Table 5 for a quick reference.

Although we have validated our theoretical findings using simulated PUF designs, we prefer to discuss our experience on the design of APUF variants in the FPGA platform. We excluded FPGA characterization results as these are different from the theoretical finding due the implementation bias in FPGA. In our discussion, we did not consider the implementation bias as it would make discussion more complex, and our focus on architectural bias would be diverted.

The APUF implementation of FPGA suffers a large implementation bias, which results in poor uniqueness metric for APUF [2, 16] even if it has good uniformity. In FPGA platform, uniqueness of APUF is even lesser than PAPUF and DAPUF. Thus, APUF implementation on FPGAs with good uniqueness is almost infeasible, but in ASIC with custom circuit layout, one can implement the APUF with good performance metrics [8, Ch. 4]. However, PAPUF and DAPUF designs are easier to implement on FPGA platform with less implementation bias, and they have less implementation effects on their performance metrics. But their performance metrics are very poor due to poor architectures, regardless of implementation platforms.

From this discussion it is evident that the it is infeasible to realize implementation bias free classic APUF on FPGA platform though it is architecturally good. On the other hand, DAPUF and PAPUF (without tuning bits) are easier to implement on FPGA with less implementation bias. Since PAPUF and DAPUF have large architectural bias, they must employ additional tuning scheme to reduce the effect of architectural bias.

In case of PAPUF design in [12], a set of additional programmable delay components are used to reduce the bias in delay differences, and those programmable delay elements are controlled by additional tuning bits, different from challenge bits. In [12], the authors suggested to use the same assignment for the tuning bits of all instances of PAPUF design, but according to the architectural bias results, it is evident that there are many PAPUF instances having different architectural bias values, and the range of bias value is also large. Thus, the idea of using the same assignment for tuning bits for all PUF instances seems to be less effective. On the other hand, if one uses different tuning bits for PAPUF instances according to its bias values, then we cannot claim the intrinsically unique behavior of PUF instances as different inputs are applied to PUF instances. We have tried with the same assignment of tuning bits for PAPUF instances, and observed that there are only a few ($< 5\%$) PAPUF instance with improved uniformity. Thus, more efficient tuning bits assignment algorithm is required to make PAPUF useful in practice.

Table 5. A comparison of ideal/simulated APUF, PAPUF and DAPUF designs

Metric	**Relation**
Arch. Bias	APUF $<$ DAPUF $\leq$ PAPUF
Uniformity	APUF $>$ DAPUF $\geq$ PAPUF
Uniqueness	APUF $>$ DAPUF $\geq$ PAPUF
Reliability	APUF $<$ DAPUF $\leq$ PAPUF
SAC Property	DAPUF $>$ APUF $>$ PAPUF
Modeling Robustness	DAPUF $\sim$ APUF $\sim$ PAPUF

Note i) "$A > B$"—PUF A is better than PUF B for a given metric and ii) "$A \sim B$"–PUF A and B exhibit a similar property.

The DAPUF design is a hybrid architecture as it borrowed the architectural properties of both the APUF and PAPUF. We can also use the tuning scheme in this context, but there is no such proposal in the literature. Since the architectural bias in the DAPUF is less compared to PAPUF, the efforts (e.g., size of tuning circuit) required to tune DAPUF seems to be less than that of PAPUF. As a future work, we try to improve DAPUF architecture as it has better SAC property and performance metrics compared to PAPUF. It is worth mentioning that tuning scheme is not feasible for APUF as bias is implementation induced, which is challenge dependent. This implies that for APUF, one needs to tuning delay difference of each challenge, and which is infeasible.

11 CONCLUSIONS

In this paper, we have introduced the notion of architectural bias, and exploited this to evaluate the architectures of APUF, PAPUF and DAPUF designs. We have shown that architectural bias is significantly less for APUF compared to DAPUF and PAPUF, and PAPUF suffers from large architectural bias. From the perspective of uniformity and uniqueness metrics, APUF architecture is superior than PAPUF and DAPUF, and DAPUF is comparatively better than PAPUF. Since more architectural bias implies good reliability, the reliability of PAPUF is better than APUF and DAPUF. Besides the performance metrics, we have compared the security metrics of APUF variants. The SAC property of DAPUF is better than APUF, and PAPUF exhibits poor SAC property among these three architectures. However, these APUF variants are vulnerable to machine learning based modeling attack. Although we have considered only the architectures of APUF design variants, but the idea of architectural bias can also be applicable to other delay PUF designs.

REFERENCES

[1] Georg T. Becker. 2015. 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)*. 535–555.

[2] Yohei Hori, Hyunho Kang, Toshihiro Katashita, Akashi Satoh, Shin-ichi Kawamura, and Kazukuni Kobara. 2014. Evaluation of Physical Unclonable Functions for 28-nm Process Field-Programmable Gate Arrays. *Journal of Information Processing (JIP)* 22, 2 (2014), 344–356. DOI: https://doi.org/10.2197/ipsjjip.22.344

[3] Zouha Cherif Jouini, Jean-Luc DANGER, and Lilian Bossuet. 2011. Performance Evaluation of Physically Unclonable Function by Delay Statistics. In *Proc. of 9th IEEE International NEWCAS conference*. 482–485.

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

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

[6] Takanori Machida, Dai Yamamoto, Mitsugu Iwamoto, and Kazuo Sakiyama. 2014. 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)*. 871–878.

[7] Takanori Machida, Dai Yamamoto, Mitsugu Iwamoto, and Kazuo Sakiyama. 2015. 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)*. 6–7. DOI: https://doi.org/10.1109/ASPDAC.2015.7058919

[8] Roel Maes. 2013. *Physically Unclonable Functions - Constructions, Properties and Applications*. Springer.

[9] Hamid Mahmoodi, Saibal Mukhopadhyay, and Kaushik Roy. 2005. Estimation of delay variations due to random-dopant fluctuations in nanoscale CMOS circuits. *IEEE Journal of Solid-State Circuits* 40, 9 (Sept 2005), 1787–1796. DOI: https://doi.org/10.1109/JSSC.2005.852164

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

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

[12] Mehrdad Majzoobi, Farinaz Koushanfar, and Srinivas Devadas. 2010. FPGA PUF using Programmable Delay Lines. In *IEEE International Workshop on Information Forensics and Security (WIFS)*. 1–6.

[13] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. 2008. Lightweight Secure PUFs. In *Proc. of the 2008 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE Press, Piscataway, NJ, USA, 670–673.

[14] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. 2008. Testing Techniques for Hardware Security. In *Proc. of IEEE International Test Conference(ITC)*. 1–10.

[15] Mehrdad Majzoobi, Masoud Rostami, Farinaz Koushanfar, Dan S. Wallach, and Srinivas Devadas. 2012. Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching. In *Proc. of IEEE Symposium on Security and Privacy Workshops*. 33–44.

[16] Sergey Morozov, Abhranil Maiti, and Patrick Schaumont. 2010. An Analysis of Delay Based PUF Implementations on FPGA. In *Reconfigurable Computing: Architectures, Tools and Applications*, Phaophak Sirisuk, Fearghal Morgan, Tarek El-Ghazawi, and Hideharu Amano (Eds.). Lecture Notes in Computer Science, Vol. 5992. Springer Berlin / Heidelberg, 382–387.

[17] Phuong Ha Nguyen, Durga Prasad Sahoo, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. 2016. Security Analysis of Arbiter PUF and Its Lightweight Compositions Under Predictability Test. *ACM Trans. Des. Autom. Electron. Syst.* 22, 2, Article 20 (Dec. 2016), 28 pages. DOI:https://doi.org/10.1145/2940326

[18] Bart Preneel, Werner Van Leekwijck, Luc Van Linden, René Govaerts, and Joos Vandewalle. 1990. Propagation Characteristics of Boolean Functions. In *Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings*. 161–173.

[19] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. 2010. Modeling Attacks on Physical Unclonable Functions. In *Proc. of 17th ACM conference on Computer and communications security (CCS)*. ACM, New York, NY, USA, 237–249.

[20] Durga Prasad Sahoo, Debdeep Mukhopadhyay, Rajat Subhra Chakraborty, and Phuong Ha Nguyen. 2016. A Multiplexer based Arbiter PUF Composition with Enhanced Reliability and Security. Cryptology ePrint Archive, Report 2016/1031, *IACR Cryptology ePrint Archive* (2016). http://eprint.iacr.org/2016/1031.

[21] G. Edward Suh and Srinivas Devadas. 2007. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Proc. of Design Automation Conference (DAC)*. ACM Press, New York, NY, USA, 9–14.

[22] Meng-Day Mandel Yu, Matthias Hiller, Jeroen Delvaux, Richard Sowell, Srinivas Devadas, and Ingrid Verbauwhede. 2016. A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication. *IEEE Transactions on Multi-Scale Computing Systems* PP, 99 (2016), 1–1. DOI:https://doi.org/10.1109/TMSCS.2016.2553027