Noise-Resilient Neural Network-Based Adversarial Attack Modeling for XOR Physical Unclonable Functions

Ahmad O. Aseeri

Department of Computer Science, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia
E-mail: a.aseeri@psau.edu.sa

Received 04 February 2020; Accepted 11 February 2020;
Publication 23 March 2020

Abstract

Authentication plays an essential role in preventing unauthorized access to data and resources on the Internet of Things. Classical security mechanisms fall short of achieving security requirements against many physical attacks due to the resource-constraint nature of IoT devices. Physical Unclonable Functions (PUFs) have been successfully used for lightweight security applications such as devices authentication and secret key generation. PUFs utilize the inevitable variation of integrated circuits during the fabrication process to produce unique responses for individual PUF, hence not reproducible even by the manufacturer itself. However, PUFs are mathematically cloneable by machine learning-based methods. XOR arbiter PUFs are one group of PUFs that can withstand existing attack methods unless exceedingly long training time and large dataset size are applied. In this paper, large-sized XOR PUFs with 64-bit and 128-bit challenges were efficiently and effectively attacked using a carefully engineered neural network-based method. Our fine-tuned neural network-based adversarial models achieve 99% prediction accuracy on noise-free datasets and as low as 96% prediction accuracy on noisy datasets, using up to 55% smaller dataset size compared to existing works known to us. Revealing such vulnerabilities is essential for PUF developers to re-evaluate existing PUF designs, hence avoiding potential risks for IoT devices.

Journal of Cyber Security and Mobility, Vol. 9_2, 331–354.
doi: 10.13052/jcsm2245-1439.926

Keywords: Physical unclonable functions, internet of things, hardware security, neural networks.

1 Introduction

The Internet of Things (IoT) has been increasingly the growing paradigm in many applied domains ranging from wearables to large industrial systems. This has raised the need for security solutions to safeguard against various attacks, especially those targeting lightweight and portable devices. The conventional cryptographic algorithms rely on secret keys stored in non-volatile memory, making them subjected to eavesdropping and side-channel attacks. Thus, providing proper security for resource-constrained IoT devices is always a challenge because traditional cryptographic approaches are resource-demanding, given the fact that resource-constrained devices inherently are of limited processing capabilities, thus making them an unsuitable choice for IoT ecosystem.

Physical unclonable functions (PUFs) have been used for low-cost security. PUFs intrinsically derive the secrets from the inevitable variations of integrated circuits, materialized during the manufacturing process, to produce unique responses for individual PUF instance, not even reproducible by the manufacturer itself. For that reason, it is almost impossible to fabricate two identical chips, even with full knowledge of the original chip's design [13]. Thus, physical unclonability defines that two PUF instances should never have identical challenge-response pair (CRPs) behavior [26], making them ideal for security purposes. PUFs have been applied as an effective, lightweight mechanism for secure authentication [19] and secret key generation [24].

Although physically unclonable, PUFs have been known to be mathematically clonable by modeling and inference-based attacks utilizing the linear additive delay nature of arbiter PUFs. Mathematical clonability implies the ability to develop malicious software that can counterfeit the trusted PUFs. To improve arbiter-based PUFs robustness against various modeling attacks, XOR arbiter PUFs, or XOR PUFs for short, were proposed by [24] to obfuscate modeling attacks from learning the behavior of PUF circuits by utilizing the power of XOR operation. For years, large-sized XOR PUFs have shown high resistance to machine learning-based modeling attacks. However, studies have successfully demonstrated the ability to break the security of XOR PUFs. Ruhrmair et al. [21] proposed the first machine learning-based models for XOR PUFs response predictions. Likewise, Tobisch et al. [25]

introduced an optimized version of Ruhrmair et al. approach to attack XOR PUFs. Although these efforts have successfully unveiled the possibility of machine learning-based methods on attacking XOR PUFs, they have some limitations. First, the proposed machine learning models by [21] fail to produce response predictions for large-sized XOR PUFs, specifically beyond 6-XOR 64-bit and 5-XOR 128-bit PUFs. Second, authors in [25] had to develop a memory-optimized parallel machine learning C language code that took three days running (or about 44 days of sequential computing time) to break one instance of 7-XOR 128-bit arbiter PUF. Recently, Aseeri et al. [4] proposed an efficient neural network method modified to handle training datasets larger than memory capacity to break XOR PUFs.

In this paper, we improve our previous work on attacking XOR PUFs presented in [4] to show the effectiveness of using the ordinary neural network to attack large-sized XOR PUFs in high prediction accuracy and shorter training time using smaller datasets size compared with existing works known to us. Besides, this paper extends our initial idea of introducing a feasible, easy-to-use, and powerful approach to attack large XOR PUFs to uncover the ability to break the security of XOR PUFs in the existence of environmental and noisy conditions that impact the reliability of PUF responses. We achieved these improvements after careful hyperparameter choices and particular neural network architecture design. The study presented in this paper reveals a serious security vulnerability, especially in large-sized XOR PUF. One key consideration in studying the ability to attack any system is to be able to identify where the vulnerabilities are. Thus, disclosing the ability to attack XOR PUFs is important to PUF designers and developers to be vigilant over these existing vulnerabilities, and hence developing and improving PUF designs to eliminate any potential risks. The paper is organized as follows. Section 2 gives a general overview of PUF technology, including the mathematical model of the PUF's challenge-response relationship that our attack is based upon. Section 3 explains the modeling attack, including the algorithm and approach we used to break the XOR PUFs. Our proposed models are evaluated practically alongside the results and discussion in Section 4. Finally, concluding remarks are given in Section 5.

2 Physical Unclonable Functions (PUFs)

PUFs are based on the fact that even though the manufacturing process is the same among different ICs, each IC is slightly different due to unavoidable manufacturing variability [13]. The idea of utilizing inherent randomness

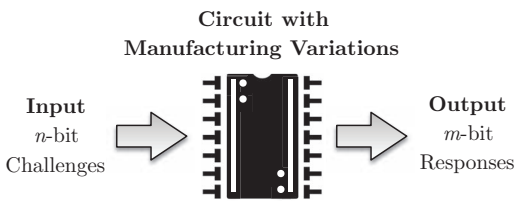

Figure 1 PUF generic diagram.

from silicon processing was proposed the first time by [18]. Physical unclonable functions leverage such physical variations to derive unique secrets from the chips instead of the standard way of storing the secret keys in digital memory. Since the introduction of PUFs, there have been plenty of PUF variations, including arbiter PUF, ring oscillator PUF, and SRAM PUF, intent to improve the underlying structure and enhance its robustness against side-channel inference attacks. As seen in Figure 1, a generic PUF diagram apply challenge-response protocol to output secrets, i.e., responses, based on physical variations of a circuit. It is an input-output mapping in the form $\Gamma : \{0,1\}^n \rightarrow \{0,1\}^m$ where the input (*challenge*) is n-bit binary vector and the output (*response*) is m-bit binary vector. A PUF circuit can receive 2^n different possible n-bit challenge vectors, each of which produces a m-bit output. As a result, silicon PUFs can generate unique challenge-response pairs (CRPs) for different integrated circuit instances [6].

PUFs are increasingly used for authentication and cryptographic key generation [13]. Classical cryptographic techniques, which mostly rely on the use of secret keys, involves two main limitations. First, secret keys that are traditionally stored in digital nonvolatile memories are vulnerable to different types of attacks such as side-channel [28] and invasive physical attacks [22]. Second, due to resource constraints, power consumption, and manufacturing costs, electronic devices may not be suitable to use secret key-based cryptographic algorithms in specific commercial scenarios [13, 20]. On the other hand, PUFs introduces a feasible solution to achieve security requirements, especially for resource-constrained devices like sensors that do not support computationally-expensive processing capabilities.

Radio-Frequency Identification technology (RFID) is the prominent application that leverages the power of PUFs for low-cost authentication to safeguard against counterfeiting attacks. That is, PUF-powered RFID tags can produce unique secrets derived from the physical randomness of a silicon chip. Because it is difficult to fabricate an identical cloned tag,

PUF-powered RFID tags are mostly the best solution for anti-counterfeiting, especially in the industrial world.

2.1 XOR Arbiter PUF

Arbiter PUF is *delay-based* silicon implementation of PUFs introduced by [11] in 2002. It is based on measuring the time delay difference between two symmetric paths. The signals race against each other simultaneously through a sequence of n stages, each of which consists of two multiplexers (MUXes) [13, 20]. At each stage, the signal transition and the delay difference between the two paths depending on the input challenge bits, which control how each MUX behaves, i.e., either straight or crossing, as shown in Figure 2. The delays introduced by the multiplexers vary with the input challenge bits where an arbiter, commonly implemented as a latch, at the end determines the possible final output either 0 or 1. That is, if the top path arrives first, then the arbiter outputs 1 or 0 otherwise.

Arbiter PUFs had known security issues because the delays were linearly added to produce the resultant response bit [13]. To overcome this issue, another variation of arbiter PUFs, named XOR arbiter PUFs, is constructed from multiple arbiter PUFs, as shown in Figure 3. A k-XOR arbiter PUF instance incorporates k arbiter components, i.e., arbiters, each of size n placed in parallel, where all of the k components take the same n-bit challenge C as an input ($C = c_1 \dots c_n$). The responses of all individual arbiter PUFs are XORed together to produce the final response r for the corresponding input challenge C. Therefore, the response of the n-stage k-XOR arbiter PUF shown in Figure 3 can be expressed as:

$$r = \bigoplus_{i=1\dots k} r_i$$

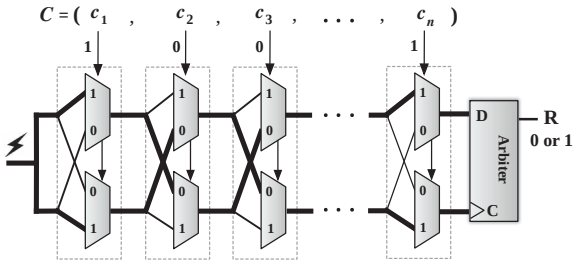

Figure 2 Arbiter PUF of n-bit length (named stages). An input challenge bit is either 1 or 0 to indicate if the signal should transit straight or crossed, respectively.

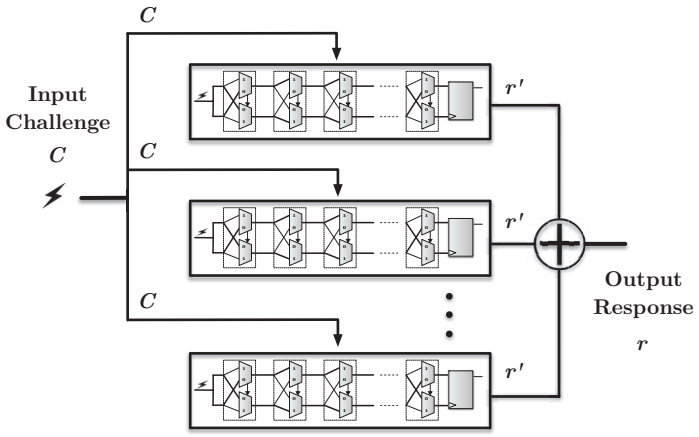

Figure 3 k-XOR Arbiter PUF. The input n-bit challenge C is fed to every arbiter. The output of all arbiters is XORed to produce the final response r.

where r_i is the internal output of the ith component arbiter PUF, and $\oplus$ indicates to XOR operation.

2.2 Modeling Arbiter PUF

The most efficient method to mathematically represent arbiter PUFs is using additive delays model of the signals at all stages of the PUF circuit. Given that each stage consists of two multiplexers, the elapsed times of the two signals arriving at the arbiter from the time the signal starts entering the first stage of the PUF are the summations of the delays incurred at each stage of the PUF. The final delay difference Δ between the upper and lower paths in n-bit arbiter PUF is the scalar multiplication between the feature vector $\vec{\Phi}$ and the transposed delay vector $\vec{\mathbf{W}}$ as follows:

$$\Delta = \vec{\Phi}\ \vec{\mathbf{W}}^T \tag{1}$$

where $\vec{\Phi}$ and $\vec{\mathbf{W}}$ are both in $n+1$ dimension. Hence, the delay difference Δ can be expressed as:

$$\Delta = \phi_1 w_1 + \phi_2 w_2 + \cdots + \phi_n w_n + w_{n+1}$$

The feature vector $\vec{\Phi}$ is derived from the challenge vector C as follows.

$$\Phi(C) = \left\{\phi_{c_1}, \phi_{c_2}, \ldots, \phi_{c_n}, 1\right\}$$

where,

$$\phi_{c_k} = \prod_{i=k}^{n} 2c_i - 1, \qquad \text{for } k = 1, \ldots, n \tag{2}$$

The parameter $\vec{\mathbf{W}}$ is a vector encoding arbiter PUF delays. Let's denote δ_i^c and δ_i^s the delays incurred from the crossed signal path and straight signal path at stage i, respectively (refer to Figure 2). Then:

$$w_i = \begin{cases} \delta_1^c - \delta_1^s, & i = 1 \\ \delta_i^c - \delta_i^s + \delta_{i-1}^c + \delta_{i-1}^s, & i = 2, ..., n \\ \delta_n^c + \delta_n^s, & i = n+1 \end{cases}$$

Since the arbiter's response is 1 if $\Delta \leq 0$ or is 0 if $\Delta > 0$, the response r can be expressed as:

$$r = sgn(\Delta) = sgn\big(\vec{\Phi}\ \vec{\mathbf{W}}^T\big) \tag{3}$$

where $sgn(\cdot)$ is the sign function. The equations above are thoroughly derived and elaborated in [4, 21].

The XOR operation increases nonlinearity in the relationship between the response r and the transformed challenges Φ. Therefore, Equation (2) plays an essential role in making modeling PUFs via machine learning method feasible. It is nothing but a linear transformation of the feature vector $\vec{\Phi}$ to transform the highly nonlinear relationship between response r and challenge vector C into a linear function of inside the sign function in equation 3. While every additional arbiter PUF increases nonlinearity and the dimensionality of the parameter space to be machine-learned by attackers [21], leading to higher resistance against machine learning attacks [17], it also increases the chip area and cost needed for silicon implementation, hence making it critical to decide the size of XOR in a PUF design.

3 Attacking Model of XOR PUFs

3.1 Mathematical Clonability

The modeling attack of PUFs and their variations are generally a non-invasive attack, meaning that if we can predict the responses of a given PUF instance in high probability, assuming a set of CRPs is drawn from the target PUF by eavesdropping or any other means, we can then break its security.

One effective yet difficult solution is the direct physical access to the target PUF to apply side-channel inference and reconfiguration of the circuits on the chip, which is not always feasible in a real scenario [10]. It has become conventionally known that attacking PUFs can be achieved by mathematical clonability through machine learning-based modeling methods. There have been several studies investigating the possibility of mathematical clonability of XOR PUFs [5, 20, 21, 25]. Those studies share almost one thought in common, XOR PUFs with large numbers of component arbiter PUFs (at least 6 PUFs) are among the PUFs family that holds high resistance against machine learning-based attacks. This is because the nonlinearity introduced by XOR operation, as well as the requirement of substantially large dataset size, to train machine learning models to increase the effort needed to break the XOR PUF instance.

The mathematical clonability of XOR PUFs is similar to arbiter PUFs in which it can be achieved by constructing a machine learning model that predicts the response of a given challenge in high accuracy and confidence. If the model prediction accuracy is high enough, it is then used for future CRP prediction. Artificial Neural Networks (ANNs) are one of the machine learning methods that have been successfully applied for supervised learning to solve complex nonlinear problems. In our previous work [4], we investigated the effectiveness of the neural network in attacking large XOR PUFs by proposing a modified neural network method to handle training datasets possibly larger than memory capacity. Since existing machine learning-based attack studies require developers to write a complex memory-optimized parallel machine learning code to be executed by parallel computing servers, the breakability of PUFs without consuming much effort to optimize the attacking code would be a more severe risk that deserves investigation. In this paper, we propose two adversarial models that achieve breaking XOR PUFs. That is, noise-free and noisy attack models.

3.2 Noise-Free Attacking Model

The essence of our attack stems from the efficient utilization of the multilayered perceptron algorithm (MLP). MLP is a class of artificial neural networks applied as a generic function approximation for supervised learning tasks. It is a sequential learning procedure that carries out the perceptron theorem in which it guarantees convergence, i.e., find the separating hyperplane in a finite number of iterations given data with linearly separable classes. As illustrated in Figure 4, MLP comprises a set of directly connected layers, each

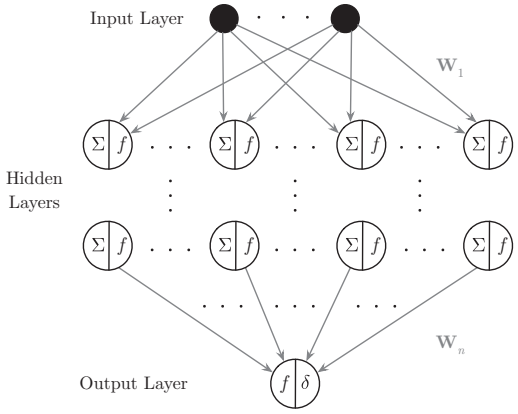

Figure 4 The generic architecture of multilayered Perceptron.

of which has a set of neurons. Theoretically, given a dataset (X, Y), where $X = \{x_1, x_2, \ldots, x_n\}$ is a set of training samples, and $Y = \{y_1, y_2, \ldots, y_n\}$ is the set of labels, the learning process in MLP is nothing but finding (approximating) the weights between the input and output layers. That is, the learning process proceeds from the input layer towards the output layer where the output of each neuron in one layer is the result of an activation function whose input is the linear combination of all outputs of neurons at the earlier layer, with the first layer being the linear combination of all scalar entries of the vector X. The weight vector for that layer denotes the coefficients of the linear combination at one layer. When the final outputs are not close enough to the target labels, the weights are tuned in such a way that minimizes the difference between the model output and the target labels.

During the process of MLP, the values of $\mathbf{W}$ shown in the Equation (1) are iteratively tuned by training the delay model (3) using an adequately large set of challenge-response pairs (CRPs). After training the model, the correct response for any challenge C can be predicted using the trained model. Therefore, the attacking method of XOR PUFs is to identify the hyperplane, by well approximation of $\mathbf{W}$, that separates the high-dimensional space of transforming challenges Φ, shown in the Equation (1), into two subspaces where $r = 0$ is in one subspace and $r = 1$ is in the other one. Additionally, we found out that using three hidden layers (excluding the input layer and the output layer) is sufficiently and efficiently enough to break XOR PUF.

Figure 5 demonstrates the high-level process of performing a machine learning-based attack on XOR PUFs. It consists of two self-explanatory phases: preprocessing phase and modeling phase. In the preprocessing phase,

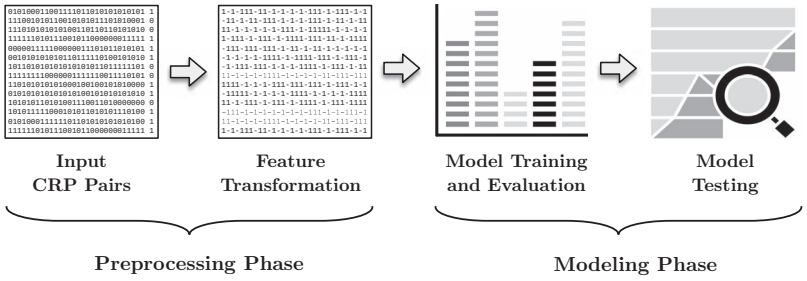

Figure 5 High-level processing view to break XOR PUFs.

the input CRPs, i.e., challenge-response pairs, are collected from an XOR PUF instance of particular xor size and stages, followed by a linear transformation applied on the input, as shown in Equation (2) in the previous section. In the modeling phase, the model in Equation (3) is trained and evaluated using an artificial neural network. After the training has completed, the final model is tested against newly random CRPs to determine how good the model is to predict responses for any CRP set.

The procedure of attacking XOR PUF is straightforward and is illustrated in Algorithm 1 for clarity. In line (2), the CRP sets are prepared, including the linear transformation of Φ shown in Equation (2), and divided into three separate subsets. In line (3), the method *construct_mlp_model()* includes our optimization of the neural network model parameters such as the architecture of the MLP and the optimization algorithm to minimize the objective function. We shall explain more about it in the next section. Finally, we store the final delay model $\mathbf{W}$ to be used for offline predictions when needed.

Algorithm 1 XOR PUF Attack

Variables Description:

- $\mathbb{X}$ is the challenge-response dataset.
- t is training time.
- γ is evaluation metrics (acc, precision, recall, F1).
- $\mathbf{W}$ is final delay model.

1: **procedure** XOR_PUF_ATTACK($\mathbb{X}$)
2: *tr_set*, *val_set*, *ts_set* ← preprocess_dataset($\mathbb{X}$)
3: *model* ← construct_mlp_model()
4: t ← train_model(*model*, *tr_set*, *val_set*)
5: $\gamma, \mathbf{W}$ ← test_model(*model*, *ts_set*)
6: **return** $\mathbf{W}, \gamma, t$
7: **end procedure**

3.3 Noisy Attacking Model

An essential property of PUFs is that the responses of a PUF instance need to be distinct between different PUFs instances but the same under various environmental conditions for the same PUF instance. Hence, a challenge is said to be unstable (or noisy) if repeated evaluations of the corresponding challenge in the same PUF instance result in different responses. In reality, the MUX-based PUF is not 100% stable. The stability of PUF responses is affected by deteriorating over time due to various sources of uncontrollable random noise such as aging, environmental variations, and power voltage [3]. In the case of delay measurements, electromagnetic interference and the supply voltage fluctuation profoundly affect the delay and can result in deceptive responses [27].

The ultimate goal of machine learning-based attack methods on XOR PUFs is to be able to predict PUF responses given randomly *unseen* challenges. This implies constructing accurate and skillful models during the modeling phase (training phase). The precision and quality of the constructed model depend on the parameter choices, and more importantly, the amount of noise exists in the training samples, i.e., challenge-response pairs. Constructing a noise-resilient machine learning-based model involves rigorous manual optimizations and careful parameter choices. To enable fair and consistent comparison in the PUF-related literature, we introduce random noise variations to the training sets to emulate the real practice of attacking XOR PUFs on deployed PUF devices when noisy conditions exist. The random noise variations introduced in our experiment can be conventionally modeled as random variables drawn from Gaussian distribution, as suggested by [1, 9, 23, 29]. To do so, we add a random Gaussian noise $G_{noise} = gaussian\ (\mu, \sigma)$ to the final delay difference Δ shown in Equation (1). Gaussian noise is nothing but a statistical noise having a probability density function equal to that of the normal distribution, i.e., the values that the noise can have is drawn from the normal distribution. Thus, the Equation (1) can be reformulated to accommodate the noise as the following:

$$\Delta = \vec{\Phi}\ \vec{\mathbf{W}}^T + G_{noise} \tag{4}$$

then, the final response r will simply be classified as:

$$r = \begin{cases} 1, & \text{if } \Delta \leq 0 \\ 0, & \text{if } \Delta > 0 \end{cases}$$

4 Experiment and Discussion

The impact of our proposed adversarial attack against XOR PUFs is evaluated by developing XOR PUF simulation written purely in Python language and launched on a computation node having 32 processors and 16GB of memory. However, the code runs only on a single CPU processor. This simulation serves as an attacking environment in which it consists of two main tasks: (1) data generation and (2) machine learning-based modeling attack employing the multilayer perceptron algorithm (MLP). The challenge-response (CRPs) data generator is capable of generating data for any XOR PUF given the desired size of XOR and the length of PUF (PUF stages). The CRPs data are randomly collected from the range $(0, 2^n)$, where n is the length of PUF. The arbiter PUF delays are drawn using a Gaussian distribution with zero mean and one unit standard deviation. For the machine learning task, we use Keras [7], an open-source library specified for a neural network that runs on top of TensorFlow. We argue that showing the success of attacking XOR PUF by conducting such a well-specified simulation has nearly a comparable impact and severity similar to possessing challenge-response datasets obtained from an FPGA implementation. This is because the multilayer perceptron (MLP) has been recognized as a universal approximator to successfully solve complex, non-linear, and real-world problems, especially when the MLP parameters are appropriately selected.

The objective of the multilayered perceptron algorithm is to reduce the loss functions that measure the inconsistency between the predicted value and the actual label. Generally speaking, the learning task via supervised learning methods works by training the constructed model using different types of optimizers such as Stochastic Gradient Descent (SGD) [14], Adaptive Moment Estimation (ADAM) [15], and Adagrad optimizer [8]. In our experiment, we have found out that ADAM optimizer, with the learning rate $\eta = 1 \times 10^{-2}$ works well in terms of the training time and the validation accuracy. The learning rate is one crucial parameter that regulates how much change can be applied to the model in response to the estimated error when the model weights are updated. Furthermore, the constructed neural network architecture consists of a varying number of hidden layers (excluding the input layer and output layer) with the different numbers of neurons (illustrated in the upcoming tables). The activation function applied by the hidden layers is the Parametric Rectified Linear Unit (pReLU), which is just a generalization of the Rectified Linear Unit (ReLU) that adaptively learns the parameters of the rectified activation units [12].

To assess the performance of the trained models as a binary classifier, we consider the following metrics: Accuracy, Precision/Recall, F-score, and Training Time. The machine learning community traditionally uses these metrics. While accuracy is a good indicator of model performance, especially for balanced data, the other metrics like F1-score and Precision/Recall provide a deeper analysis of the model behavior. More useful interpretation of these metrics is given by [2, 16]. For the sake of completeness, we explain these metrics below using a similar interpretation from this paper [2]. Let C be a set of challenge vectors. Let $f(c)$ be the output of the predictive model given the challenge $c \in C$, and y_c is the ground truth label (i.e., the actual label). Let TP, TN, FP, FN denote to True Positive, True Negative, False Positive, and False Negative, respectively. Then:

$$\begin{aligned}
TP(f,y) &= |\{f(c) = 1 \cap y_c = 1 | c \in C\}| \\
TN(f,y) &= |\{f(c) = 0 \cap y_c = 0 | c \in C\}| \\
FP(f,y) &= |\{f(c) = 1 \cap y_c = 0 | c \in C\}| \\
FN(f,y) &= |\{f(c) = 0 \cap y_c = 1 | c \in C\}| \\
Acc(f,y) &= \frac{TP(f,y) + TN(f,y)}{|C|} = \frac{|\{y_c = f(c) | c \in C\}|}{|C|} \\
Pre(f,y) &= \frac{TP(f,y)}{TP(f,y) + FP(f,y)} \\
Rec(f,y) &= \frac{TP(f,y)}{TP(f,y) + FN(f,y)} \\
F1(f,y) &= 2 \times \frac{Pre(f,y) \cdot Rec(f,y)}{Pre(f,y) + Rec(f,y)}
\end{aligned} \tag{5}$$

For each k-XOR PUF, we run the experiment 5 times and report the *average* result for each metric. For the machine learning process, the input dataset, denoted by $\mathbb{X}$ by Algorithm 1, is preprocessed by the method (preprocess_dataset) and distributed into three sets: 70% of the data are used for training, 10% is used for evaluation, and the remaining 20% is reserved for testing. We emphasize that the testing data are *unseen* during the training process to help us evaluate the generalization of our prediction model on new data. In addition, we compare the results of our constructed attack models, in which we show the size of training data and training time are reduced, against two previous research efforts related to our work: (i) the first proposed model

on breaking the security of XOR PUF introduced by Ruhrmair et al. [21], and (ii) our previous work in [4].

Table 1 (see next page) illustrates the results of *noise-free* attack models on XOR PUFs for two stage lengths 64-bit and 128-bit shown in the Tables 1(a) and 1(b), respectively. The first column $\oplus$ indicates to the XOR size, which is nothing but the number of parallel stages, i.e., arbiter PUFs, that their responses are being XORed (see the Figure 3 for illustration). In the column "Model", we compare three models (A, B, C), each of which is meant to denote specific work. Model A is the model results introduced by Ruhrmair et al. [21] as the first machine learning-based attempt to break XOR PUF. Model B indicates the work by Aseeri et al. [4], and it achieved better results than model A via the neural network approach. Model C indicates the results of the proposed models in this paper, emphasized in bold, where we show the improved results over the models A and B in terms of the number and the size of training data as well as the training time. The column "Training Set" lists the *minimum* the number of training datasets (and their sizes) needed to attain successful training of the models. The column "Memory Usage" lists the amount of memory utilized by the training process of our model using the Linux command *htop*. Note that the empty cells, filled by (N/A), imply no answers provided for the corresponding metric by the models A and B. Due to space limit, we direct the reader to the Appendix (Subsection A.1) to see the MLP hyperparameters choices to achieve the results in Table 1.

From Table 1, it can be realized that while all three models attain almost similar prediction accuracies in attacking XOR PUFs, our proposed model C is able to attain a solid predictive power using smaller sets of training data compared to model B. Specifically, for 64-bit stage, the model C uses in average 40% smaller training data size than the model B and, for 128-bit stage, in average 55% smaller training data size than the model B. Keep in mind that comparing between the models B and C while excluding the model A is rational here because the models B and C use the same machine learning method, the neural network, while the model A uses different machine learning method, the logistic regression. Furthermore, model C achieves faster training time compared to the other two models A and B, which shows how good our precise selections of the multilayered perceptron parameters can positively affect the performance as it makes the gradient descent find the optimal weights very quickly. Moreover, we demonstrate the training loss curves from our attack model C in Figures 6 and 7 for different XOR sizes. In these plots, the vertical axis represents the loss values, and the horizontal axis represents the number of epochs the model conducted. As seen, our

Table 1 Results of *noise-free* attack models on XOR PUFs for two PUF stages lengths: (a) 64-bit and (b) 128-bit. The first column $\oplus$ indicates to the XOR size while the second one $\mathbb{M}$ indicates to the model type for comparison

$\oplus$	$\mathbb{M}$	Training Set # CRPs (Size)	Evaluation Metrics Accuracy	Evaluation Metrics F1 Score	Training Time	Memory Usage
(a) **When PUF stage length is 64-bit**						
4	A	12×10^3	99%	N/A	3.32 min	N/A
	B	0.4×10^6 (0.2 GB)	98.42%	N/A	19.2 sec	N/A
	C	**200×10^3 (12.8 MB)**	**99.11%**	**99.02%**	**29.21 sec**	**52 MB**
5	A	80×10^3	99%	N/A	2.08 hrs	N/A
	B	0.8×10^6 (0.4 GB)	98.55%	N/A	58 sec	N/A
	C	**500×10^3 (32.5 MB)**	**99.09%**	**99.09%**	**1.41 min**	**130 MB**
6	A	200×10^3	99%	N/A	31.01 hrs	N/A
	B	2×10^6 (1 GB)	99.15%	N/A	7.4 min	N/A
	C	**900×10^3 (58.5 MB)**	**99.01%**	**99.01%**	**4.54 min**	**234 MB**
7	A	N/A	N/A	N/A	N/A	N/A
	B	5×10^6 (2.5 GB)	99.21%	N/A	11.8 min	N/A
	C	**4×10^6 (260 MB)**	**99.07%**	**99.07%**	**9.26 min**	**1.04 GB**
(b) **When PUF stage length is 128-bit**						
4	A	24×10^3	99%	N/A	2.52 hrs	N/A
	B	0.8×10^6 (0.8 GB)	98.46%	N/A	1.32 min	N/A
	C	**400×10^3 (51.6 MB)**	**99.04%**	**99.04%**	**1.2 min**	**206.4 MB**
5	A	500×10^3	99%	N/A	16.36 hrs	N/A
	B	3×10^6 (3 GB)	98.55%	N/A	5 min	N/A
	C	**2.5×10^6 (322 MB)**	**99.06%**	**99.06%**	**4.39 min**	**1.3 GB**
6	A	N/A	N/A	N/A	N/A	N/A
	B	20×10^6 (20 GB)	99.03%	N/A	19 min	N/A
	C	**4.5×10^6 (580.5 MB)**	**99.06%**	**99.06%**	**13.37 min**	**2.3 GB**
7	A	N/A	N/A	N/A	N/A	N/A
	B	40×10^6 (40 GB)	99.00%	N/A	1.5 hrs	N/A
	C	**10×10^6 (1.29 GB)**	**99.11%**	**99.11%**	**25.8 min**	**5.16 GB**

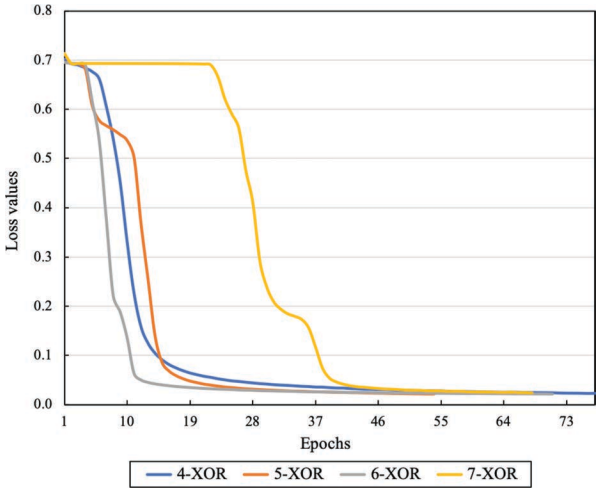

Figure 6 Training loss curves of the noise-free attack models applied on different XOR PUF sizes when stage length is 64 bit. For each XOR size, the curve is averaged over 5 runs.

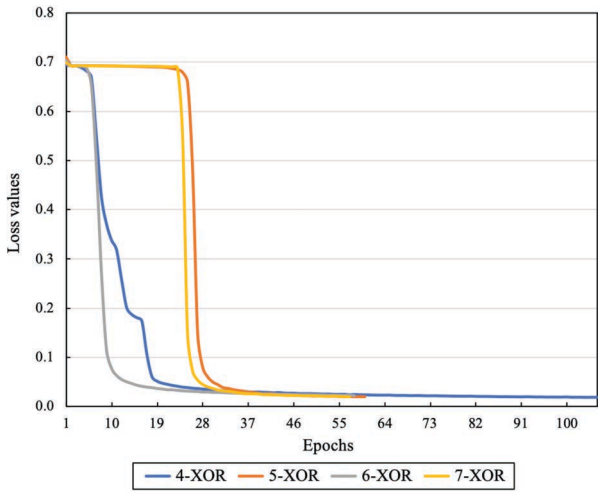

Figure 7 Training loss curves of the noise-free attack models applied on different XOR PUF sizes when stage length is 128 bit. For each XOR size, the curve is averaged over 5 runs.

models reach the lowest possible prediction errors, manifesting the ability to predict the correct responses of any given challenge set. Note that although we carried out experiments up to 7-XOR PUFs only, we anticipate that it is unlikely that this many PUFs will be applied in practice due to the

Table 2 Results of the proposed noisy attack models applied on 7-XOR 64 bit PUFs using different noise values. The structure of MLP consists of three hidden layers each of which has 128 neurons. The batch size is set to be 100×10^3

Noise Value	Training Set	Evaluation Metrics		Training	Memory
G_{noise}	# CRPs (Size)	Accuracy	F1 Score	Time	Usage
0.1	6×10^6 (390 MB)	98.31%	98.31%	6.68 min	1.56 GB
0.2	7×10^6 (455 MB)	98.22%	98.22%	8.66 min	1.82 GB
0.3	7×10^6 (455 MB)	97.35%	97.34%	7.59 min	1.80 GB
0.4	8×10^6 (520 MB)	97.25%	97.25%	9.28 min	2.09 GB
0.5	8×10^6 (520 MB)	96.83%	98.83%	16.26 min	2.08 GB

Table 3 Results of the proposed noisy attack models applied on XOR Arbiter PUFs when noise value $G_{noise} = 0.5$. The first column $\mathbb{L}$ indicates to the stage length while the second column $\oplus$ indicates to the XOR size.

		Training Set	Evaluation Metrics		Training	Memory
$\mathbb{L}$	$\oplus$	# CRPs (Size)	Accuracy	F1 Score	Time	Usage
64 bits	4	300×10^3 (19.5 MB)	97.78%	97.81%	14.61 sec	78 MB
	5	600×10^3 (39 MB)	97.35%	97.33%	39.34 sec	156 MB
	6	1×10^6 (65 MB)	97.05%	97.05%	4.13 min	260 MB
	7	8×10^6 (520 MB)	96.83%	98.83%	16.26 min	2.08 GB
128 bits	4	500×10^3 (64.5 MB)	97.96%	97.94%	18 sec	258 MB
	5	3×10^6 (387 MB)	97.41%	97.42%	3.83 min	1.54 GB
	6	6×10^6 (774 MB)	97.78%	97.78%	12.34 min	3.09 GB

expected response unreliability as well as the overhead and cost associated with manufacturing the large size of XOR PUFs.

Tables 2 and 3 illustrate the results of attack models using noisy data. The Table 2 presents the results of noisy attack models applied on 7-XOR 64-bit PUFs using different noise values G_{noise} ranging from 0.1 to 0.5, as suggested by [1, 29]. In general, modeling noisy data has always been a challenge because it may lead to poor generalization. As noticed in both tables, it is still considerably impressive to achieve as low as 96% prediction accuracy in relatively faster training time. This implies that our careful choices of multilayered perceptron parameters alongside the problem-tailored design of MLP play a significant role in achieving these successful results. Besides, we plot the training loss curves of our noisy attack models presented in Table 2 in Figure 8. As shown in this plot, the prediction errors decrease, enabling the resulted model to be able to predict the responses of any given noisy

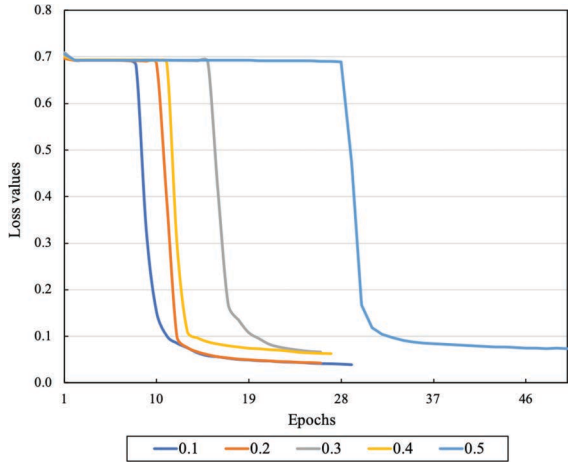

Figure 8 Training loss curves of the proposed noisy attack models applied on 7-XOR 64-bit PUFs using different noise values. For each noise value, the curve is averaged over the total of 5 runs.

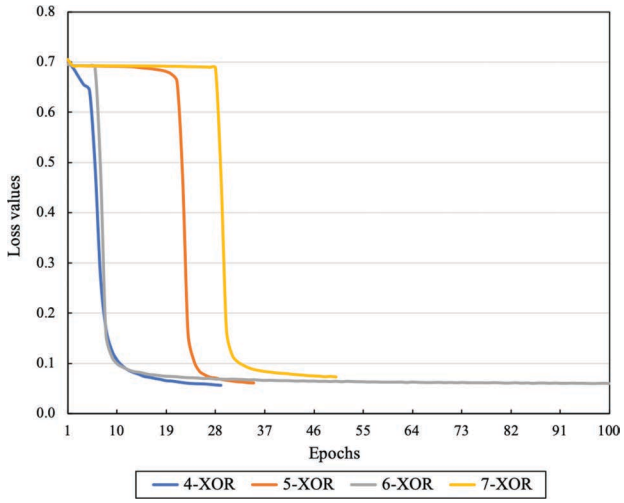

Figure 9 Training loss curves when the noise value is $G_{noise} = 0.5$ for PUF stage length 64 bit. For each XOR size, the curve is averaged over 5 runs.

challenge input. Similarly, Table 3 shows the results of noisy attack models applied on different XOR PUFs sizes when noise value is $G_{noise} = 0.5$. The training loss curves of this table are plotted in Figures 9 and 10. As shown, the model can achieve at least 96% prediction accuracies given noisy data.

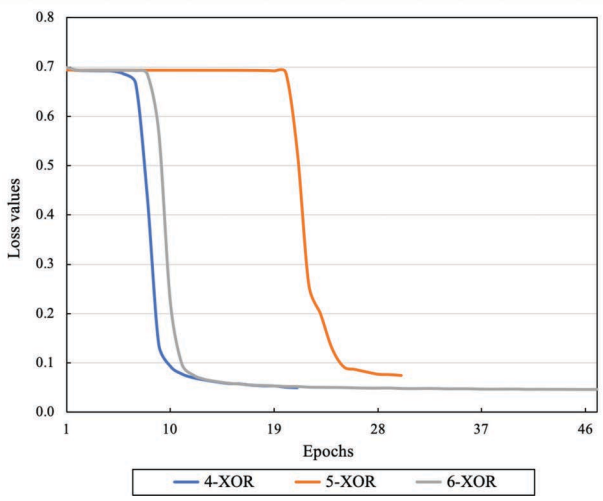

Figure 10 Training loss curves when the noise value is $G_{noise} = 0.5$ for PUF stage length 128 bit. For each XOR size, the curve is averaged over 5 runs.

Due to space limit, we direct the reader to the Appendix (Subsection A.2) to see the MLP hyperparameters choices to achieve the results in Table 3.

5 Conclusion

In this paper, we efficiently and effectively showed the possibility of breaking the security of large-sized XOR PUFs in high prediction accuracy and shorter training time using smaller datasets size compared with other existing works known to us. Our attack process employs the multilayered perceptron method to construct predictive models whose architecture is well constructed, and the parameters were rigorously crafted to achieve successful response prediction of any given XOR PUF. Additionally, our proposed method is noise-resilient, making the attack more severe. Revealing the ability to attack XOR PUFs is important to PUF designers and developers to be aware of existing vulnerabilities, and therefore improving the PUF designs to resist against potential risks.

Acknowledgement

This Publication was supported by the Deanship of Scientific Research at Prince Sattam bin Abdulaziz University.

Appendix

A.1 MLP Hyperparamters Choices: Noise-Free Attack Models

Table A.1 shows the hyperparameter choices used by the MLP to construct *noise-free* attack models on XOR Arbiter PUFs for two PUF stage lengths: 64 bit and 128 bit. The column "Hidden Layers" illustrates the size of the MLP's hidden layers, excluding the input and the output, and the number of neurons at each layer. For instance, (16, 16) means two hidden layers, each of which includes 16 neurons. The other columns are self-explanatory.

A.2 MLP Hyperparamters Choices: The Proposed Noisy Attack Models

Table A.2 shows the hyperparameter choices used by the MLP to construct the proposed *noisy* attack models applied on XOR PUFs when noise value $G_{noise} = 0.5$ for two PUF stage lengths: 64 bit and 128 bit. Again, the column "Hidden Layers" illustrates the size of the MLP's hidden layers, excluding the input and the output, and the number of neurons at each layer. For instance, (16, 16) means two hidden layers, each of which includes 16 neurons. The other columns are self-explanatory.

Table A.1 Hyperparameter choices for *noise-free* attack models on XOR Arbiter PUFs for two PUF stage lengths: 64 bit and 128 bit. Refer to Table 1 for the results

Stage	XOR Size	Optimizer	Batch Size ($\times 10^3$)	Hidden Layers (# Neurons)	Activation Function	Learning Rate
64 bit	4	ADAM [15]	10	(16, 16)	pReLU [12]	1×10^{-2}
	5		10	(32, 32)		
	6		10	(64, 64)		
	7		50	(128, 128, 128)		
128 bit	4	ADAM [15]	10	(16, 16)	pReLU [12]	1×10^{-2}
	5		30	(128, 128, 128)		
	6		10	(128, 128, 128)		
	7		100	(128, 128, 128)		

Table A.2 Hyperparameter choices for the proposed *noisy* attack models applied on XOR PUFs when noise value $G_{noise} = 0.5$ for two PUF stage lengths: 64 bit and 128 bit. Refer to Table 3 for the results

Stage	XOR Size	Optimizer	Batch Size ($\times 10^3$)	Hidden Layers (# Neurons)	Activation Function	Learning Rate
64 bit	4	ADAM [15]	10	(16, 16)	pReLU [12]	1×10^{-2}
	5		10	(32, 32)		
	6		10	(64, 64, 64)		
	7		100	(128, 128, 128)		
128 bit	4	ADAM [15]	10	(16, 16)	pReLU [12]	1×10^{-2}
	5		10	(128, 128, 128)		
	6		10	(128, 128, 128)		

References

[1] Yousra Alkabani, Farinaz Koushanfar, Negar Kiyavash, and Miodrag Potkonjak. Trusted integrated circuits: A nondestructive hidden characteristics extraction approach. In *International Workshop on Information Hiding*, pages 102–117. Springer, 2008.

[2] Mohammed Saeed Alkatheiri and Yu Zhuang. Towards fast and accurate machine learning attacks of feed-forward arbiter pufs. In *Dependable and Secure Computing, 2017 IEEE Conference on*, pages 181–187. IEEE, 2017.

[3] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Francois-Xavier Standaert, and Christian Wachsmann. A formalization of the security features of physical functions. In *2011 IEEE Symposium on Security and Privacy*, pages 397–412. IEEE, 2011.

[4] Ahmad O. Aseeri, Yu Zhuang, and Mohammed Saeed Alkatheiri. A machine learning-based security vulnerability study on xor pufs for resource-constraint internet of things. In *2018 IEEE International Congress on Internet of Things (ICIOT)*, pages 49–56. IEEE, 2018.

[5] Georg T. Becker. The gap between promise and reality: On the insecurity of xor arbiter pufs. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 535–555. Springer, 2015.

[6] Urbi Chatterjee, Rajat Subhra Chakraborty, Hitesh Kapoor, and Debdeep Mukhopadhyay. Theory and application of delay constraints in arbiter puf. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):10, 2016.

[7] François Chollet et al. Keras, 2015.
[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
[9] Jean-Pierre Seifert Fatemeh Ganji, and Shahin Tajik. A fourier analysis based attack against physically unclonable functions. *Springer*, 2018.
[10] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. Why attackers win: on the learnability of xor arbiter pufs. In *International Conference on Trust and Trustworthy Computing*, pages 22–39. Springer, 2015.
[11] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 148–160. ACM, 2002.
[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.
[13] Charles Herder, Meng-Day Yu, Farinaz Koushanfar, and Srinivas Devadas. Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, 102(8):1126–1141, 2014.
[14] Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.
[15] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
[16] Oluwasanmi O. Koyejo, Nagarajan Natarajan, Pradeep K. Ravikumar, and Inderjit S. Dhillon. Consistent binary classification with generalized performance metrics. In *Advances in Neural Information Processing Systems*, pages 2744–2752, 2014.
[17] Daihyun Lim, Jae W. Lee, Blaise Gassend, G. Edward Suh, Marten Van Dijk, and Srinivas Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205, 2005.
[18] Keith Lofstrom, W. Robert Daasch, and Donald Taylor. Ic identification circuit using device mismatch. In *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No. 00CH37056)*, pages 372–373. IEEE, 2000.
[19] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.

[20] Ulrich Ruhrmair and Daniel E Holcomb. Pufs at a glance. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[21] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 237–249. ACM, 2010.

[22] Sergei Petrovich Skorobogatov. *Semi-invasive Attacks: A New Approach to Hardware Security Analysis*. PhD thesis, University of Cambridge Ph. D. dissertation, 2005.

[23] Ashish Srivastava, Dennis Sylvester, and David Blaauw. *Statistical Analysis and Optimization for VLSI: Timing and Power*. Springer Science & Business Media, 2006.

[24] Suh G. Edward and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th Annual Design Automation Conference*, pages 9–14. ACM, 2007.

[25] Johannes Tobisch and Georg T. Becker. On the scaling of machine learning attacks on pufs with application to noise bifurcation. In *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 17–31. Springer, 2015.

[26] Sying-Jyan Wang, Yu-Shen Chen, and Katherine Shu-Min Li. Adversarial attack against modeling attack on pufs. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 138. ACM, 2019.

[27] Xiaoxiao Wang and Mohammad Tehranipoor. Novel physical unclonable function with process and environmental variations. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 1065–1070. IEEE, 2010.

[28] Yuval Yarom and Katrina Falkner. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium*, pages 719–732, 2014.

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

Biography

Ahmad O. Aseeri is currently an Assistant Professor at the Department of Computer Science, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Saudi Arabia. He received his B.Ed. in Computing from King Saud University, Saudi Arabia; M.Sc. in Computer Science from University of Wisconsin-Milwaukee, United States; and Ph.D. in Computer Science from Texas Tech University, United States. Dr. Aseeri's research interest is in the field of Artificial Intelligence (AI), having the main focus in the area of (1) Deep Learning: with application to neural network-based risk analysis, natural language processing (NLP), and computer vision, (2) Data Mining: with application to clustering techniques including bisecting K-means clustering (BKM), limited-iteration bisecting K-means (LIBKM), and memory-aware clustering algorithms. He also has a research interest in applied deep learning for medical applications.