# Deep Learning based Model Building Attacks on Arbiter PUF Compositions

Pranesh Santikellur, Aritra Bhattacharyay, and Rajat Subhra Chakraborty, *Senior Member, IEEE*

***Abstract*—Robustness to modeling attacks is an important requirement for PUF circuits. Several reported Arbiter PUF compositions have resisted modeling attacks. and often require huge computational resources for successful modeling. In this paper we present deep feedforward neural network based modeling attack on 64-bit and 128-bit Arbiter PUF (APUF), and several other PUFs composed of Arbiter PUFs, namely, XOR APUF, *Lightweight Secure PUF* (LSPUF), Multiplexer PUF (MPUF) and its variants (cMPUF and rMPUF), and the recently proposed Interpose PUF (IPUF, up to the (4,4)-IPUF configuration). The technique requires no auxiliary information (e.g. side-channel information or reliability information), while employing deep neural networks of relatively low structural complexity to achieve very high modeling accuracy at low computational overhead (compared to previously proposed approaches), and is reasonably robust to error-inflicted training dataset.**

***Index Terms*—Arbiter PUF compositions, Deep Learning, Modeling attacks, Physically Unclonable Functions.**

## I. Introduction

In the context of semiconductor technology, *Physically Unclonable Functions* (PUFs) are electronic circuits that manifest the impact of nanoscale process variation induced randomness of modern semiconductor manufacturing technology [1], [2]. An ideal PUF should be unclonable (by manufacturing or by mathematical modeling), and each individual PUF instance should have unique characteristics, which should clearly distinguish it from other instances of the same PUF family. Typically, this unique characteristic is in the form of the truth table of the PUF instance, and an $n$-bit input, $m$-bit output PUF instance can be abstractly viewed to be a Boolean function $f: \{0,1\}^n \rightarrow \{0,1\}^m$. This Boolean mapping should ideally be unique to each PUF instance, and should remain constant over time in spite of variations in environmental conditions and the electrical noise environment (i.e., the PUF should be perfectly reliable). PUFs are useful hardware security primitives, and have been proposed to be used in a plethora of applications, ranging remote hardware attestation to lightweight authentication protocol development [1]–[6]. An $n$-bit input to a PUF instance, and the corresponding $m$-bit output generated by the PUF instance, together constitute a "Challenge-Response Pair" (CRP).

P. Santikellur and R. S. Chakraborty are with the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, West Bengal, India 721302. E-mail: pranesh.sklr@iitkgp.ac.in, rschakraborty@cse.iitkgp.ac.in.

A. Bhattacharyay is with the School of Computer Engineering, Kalinga Institute of Industrial Technology, Bhubaneswar, Odisha, India 751024. E-mail: aritrab97@gmail.com.

Pranesh Santikellur is funded by Intel Corporation.

Unfortunately, the promise of PUFs have largely remained unfulfilled in practice, due to the gap between the ideal characteristics of a hypothetical PUF and the real characteristics of practically implementable PUFs. Often PUF instances are found to be quite similar in characteristics to each other ("poor uniqueness"), and often their truth tables are not constant over time of when subjected to environmental variations ("poor reliability"). Also, PUFs have been shown to be susceptible to a plethora of attacks, ranging from direct physical cloning [7] and laser based fault attacks [8], to mathematical modeling attacks (primarily machine learning based) [1], [9]–[15]. Currently, the modeling attacks are considered to be the greatest threat against PUF implementations, and wide-ranging intensive research is currently underway to develop newer attacks, as well as PUF designs robust to known attacks. In these attacks, typically a small fraction of the CRP dataset of a particular PUF instance is made available to the adversary, based on which she builds an accurate computational model of the PUF instance, capable of predicting the response for an arbitrary challenge with high probability of success.

Among silicon PUF implementations, the *Arbiter PUF* (APUF) is perhaps the most widely studied PUF variant, being analyzed and implemented since the earliest days of research on PUFs [1]. Although stand-alone APUF by itself is not deployable because of its severe vulnerability to modeling attacks [1], [10], APUFs still have unique advantages regarding simplicity and regularity in design and operating principles, low hardware overhead, and being amenable to be VLSI implementation. Hence, APUF continues to be used as a building block for more secure APUF compositions, the most common among which are: *XORPUF*, *Lightweight Secure PUF* (LSPUF) [16], *Multiplexer PUF* (MPUF) and its variants cMPUF and rMPUF [4], and the recently proposed *Interpose PUF* (IPUF) [17]. Successful attacks have been reported previously against even these variants in recent years, but many of them require access to additional information, e.g. side channel information [11], reliability characterization information [18], etc. Some other proposed attacks on these variants which do not require access to auxiliary information have also been proposed, but they are unable to reach the high levels of accuracy typically reached by machine learning based techniques, and are only successful in a "cryptanalytic" sense [19], meaning the probability of successful prediction of the response corresponding to an arbitrary challenge is greater than $\frac{1}{2}$). Sometimes, these attacks are successful in predicting the response for a challenge only if the responses of some related challenges are known [20].

One of the first deep learning based attack on APUF

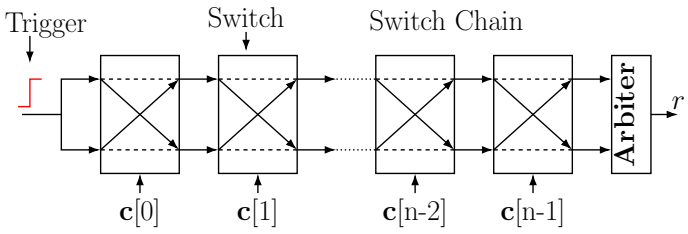


Fig. 1. An $n$-bit Arbiter PUF (APUF) [1].

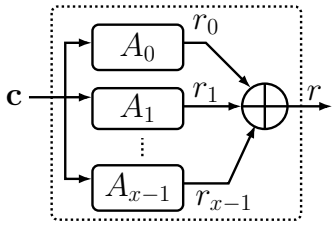


Fig. 2. An $x$-XOR APUF.

using pure black-box technique was reported in [21], but it obtained much poorer modeling accuracy ($\sim 58\%$) compared to previously reported ML algorithms. Another deep learning based attack [22] uses de-noising autoencoders for evaluating the security of 3-XOR APUF and *Double Arbiter PUF* (DA-PUF), a variant of APUF with enhanced uniqueness. Because de-noising autoencoders were used during the pre-training phase, the reported computation time taken is significantly high than ML algorithms. Also, they did not try the attacks on challenging APUF compositions such as 5-XOR APUF or LSPUF. Details about the operating principles of these PUF variants, and existing attacks on them are provided in Section II.

In this paper we have described deep learning based successful attacks on the above APUF compositions. Note that machine learning of XOR-type functions is a classic computational problem, and has been widely studied for several decades. Relatively recently, it has been suggested [23] based on other theoretical advancements [24] that deep feedforward neural networks with the *Rectified Linear Unit* (ReLU) as the activation function in the hidden layers are possibly good choices for learning XOR networks. This was an important motivation for our efforts of applying deep learning (in particular, deep feedforward networks) in modeling of APUF compositions, many of which are based on XOR-based combinations of APUF responses. Also, MPUF and its variants are similar to XOR PUFs with respect to their Boolean functional representation. A recent attack [25] applies *Artificial Neural Network* (ANN) to model XOR APUF, and obtains very promising results for 8-XOR APUF (64–bit challenge size) and 7-XOR APUF (128-bit challenge size). However, we faced issues trying to replicate the experiments described in [25], as detailed in Section IV.

To summarize, in this paper, we make the following contributions:

- To the best of our knowledge, **this is the first successful application of Deep Learning in attacking a wide variety of APUF compositions**, including challenging ones such as 6-XORPUF, LSPUF (up to each output being 6-XOR), and IPUF (up to the (4,4)-IPUF configuration), for 64-bit and 128-bit challenge sizes.
- The proposed technique uses only parity-vectors derived from the challenge as input features, and it does not require any auxiliary information such as side-channel information or reliability-related information. The proposed attack also does not require the application of external stimulus to modify the functionality or induce fault in the PUF. According to the categorization of PUF modeling attacks proposed in [17], the proposed attack can be categorized as "classical derivative based white box attack", since:
  - It requires to know the structure of the PUFs under attack ("white box").
  - The proposed neural network uses gradient based numerical optimization methods ("derivative based").
  - It only requires CRPs for launching the attack ("classic attack").
- The technique achieves high modeling accuracy, at acceptable computational overhead, with all computations being carried on a single high-end workstation. **Our goal is to demonstrate that deep learning is capable of modeling practical PUF variants which have resisted standalone machine learning based modeling till date, using an ordinary computational infrastructure (i.e. without using high-performance clusters, etc., in contrast to [26].**
- No explicit "feature engineering" is required to design custom input features for achieving high accuracy. The technique is reasonably robust to errors in the training dataset.
- Lastly, we have made the dataset and machine learning software code used in our experiments available online[1], for the benefit of the research community and further extension of the work.

The rest of the paper is organized as follows. Section II presents background information on the PUF circuits considered, with previously reported attacks for each variant. We describe our proposed attack and architecture of the deep neural network used by us in Section III. Results are discussed in Section IV. We conclude in Section V with directions of future research.

## II. Background: APUF and APUF Compositions

### A. Arbiter PUF (APUF)

The APUF consists of a cascade of several structurally identical two-port delay stages, with an arbiter (usually a latch) at the end of the cascade. For each delay stage (say, the $i$-th stage), which are path-swapping switches, connectivity from the input to output ports is controlled by a control bit (the $i$-th challenge bit $c[i]$). The two input ports of the first stage are shorted, and an input pulse is allowed to propagate along the two delay paths. The exact paths followed by the two signals

[1] https://github.com/Praneshss/Modeling_of_APUF_Compositions

along the two paths depends on the challenge applied. Because of process variation effects, in spite of (ideally) identical layout, the signals along the two paths reach the arbiter after slightly different delays, resulting in either logic-0 or logic-1 output of the arbiter. Fig. 1 shows an $n$-bit APUF circuit. For each $n$-bit challenge $\mathbf{c} \in \{0,1\}^n$, it can be proved that the response of an APUF to for a given challenge $\mathbf{c}$ is given by:

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

and the delay difference at the end of the cascaded switch stages, $\Delta_{\mathbf{c}}$ is given by: $\Delta_{\mathbf{c}} = \mathbf{w}^T \mathbf{\Phi}$. Here, $\mathbf{w}$ is known as the *weight vector*, and is a $(n+1)$-dimensional vector of real numbers, with components dependent on the path delay, and $\mathbf{\Phi}$ is the *parity vector* derivable from the given challenge $\mathbf{c}$, whose components are given by: $\mathbf{\Phi}[n] = 1$ and $\mathbf{\Phi}[i] = \prod_{j=i}^{n-1}(1-2c[j])$, $i = 0, 1, \cdots n-1$. Hence, successful modeling of an APUF comes down to accurately predicting the weight vector $\vec{w}$. This can be achieved quite successfully by several machine learning algorithms, viz., *Support Vector Machine* (SVM) [1], [10], *Logistic Regression* (LR) [10], *Probably Approximately Correct Learning* (PAC Learning) [15], etc. APUFs can also be attacked in a limited way, taking advantage of their poor *Strict Avalanche Criterion* (SAC) property, to accurately predict the response for challenges, if responses of similar challenges are known [20].

### B. XOR Arbiter PUF (XOR APUF)

To increase the robustness of APUF to machine learning based modeling attacks, it was proposed to XOR the output of multiple (say, $x$) APUFs, all being input the same challenge [3]. Fig. 2 shows the structure of an $x$-XOR APUF. Increasing the value of $x$ results in exponential increase in training data and training time requirement in classical machine learning attacks [10]. The same work mathematically modeled the x-XOR PUF and expressed it as:

$$response_{XOR} = \prod_{i=1}^{x} sign(\vec{w}_i^T \vec{\Phi}_i) = sign(\prod_{i=1}^{x} \vec{w}_i^T \vec{\Phi}_i) \quad (2)$$

where $\vec{\Phi}$ is the "parity vector" corresponding to the applied (bipolar-encoded) challenge vector. In [10] it was reported that modeling attack on $x$-XOR APUF is infeasible if $x \geq 6$, although later careful implementation of LR was able to break $x$-XOR APUF for $x \leq 9$ [26], albeit employing substantial computational resources (e.g. a cluster with 1 TB of main memory), and large amount of CRP data (e.g. 350 million CRPs for the 9-XOR APUF). $x$-XOR APUFs have also been attacked successfully by PAC learning for $x \leq 5$ [14]. Power and timing side channel information dependent machine learning attack on $x$-XOR APUF was reported in [11] for up to $x = 16$. A major hindrance with large XOR APUFs is that the reliability of the $x$-XOR APUF decreases rapidly with an increase in the value of $x$. In fact, a powerful attack on $x$-XOR APUF, taking advantage of its imperfect reliability characterization information, where the attack time complexity increases only linearly with $x$ was reported in [18]. Besides reliability characterization information, this attack employs *Evolution Strategies* (ES) as the ML modeling algorithm; however, this attack requires the adversary to characterize a given XOR APUF instance multiple times for the same set of challenges.

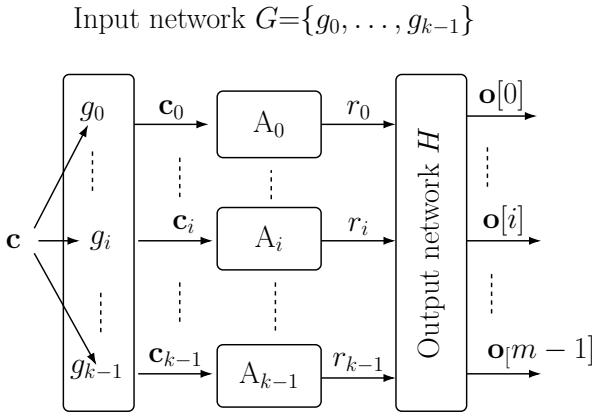


Fig. 3. An $m$-output Lightweight Secure PUF (LSPUF) [16].

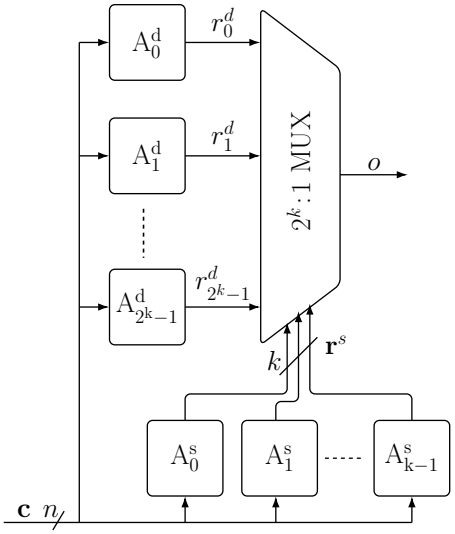


Fig. 4. Architecture of an $(n,k)$-MPUF [4].

### C. Lightweight Secure PUF (LSPUF)

The LSPUF was designed to provide an security enhancement over the XOR APUF [16]. It is a multi-bit output PUF (say, $m$ output bits), where each output bit is generated by XOR-ing a fixed number (say, $x$) of outputs of a set of APUFs (say, $k$ total APUFs), according to the formula:

$$\mathbf{o}[i] = \bigoplus_{j=0}^{x-1} r_{((i+s+j) \mod k)} \quad (3)$$

where $o[i]$ is the $i$-th output bit, $i = 0, 1, \cdots m-1$; $r_j$ denote the $j$-th output bit of the $j$-th APUF; $s$ is a circular shift parameter which determines how the $x$ APUF output values are chosen, and $x < k$. A major architectural difference

with the XOR APUF is that now every constituent APUF receives a different challenge vector, being derived from the input challenge $\vec{c}$ by an input logic network $G$. This helps to enhance the security of the circuit with respect to the XOR APUF [16]. However, each output bit of the LSPUF still suffers from the reliability issue of XOR APUFs, hence the value of $x$ cannot be made arbitrarily large in practice. Fig. 3 shows an example LSPUF with $m$-output bits, and $k$ constituent APUFs. The LSPUF (each individual output bit) was attacked successfully through LR (up to each output being 5-XOR) [10], cryptanalysis-assisted machine learning [19], and like the XOR APUF, through power and timing side channel supplemented machine learning attacks (up to each output being 16-XOR) [11].

### D. Multiplexer PUF (MPUF) and Its Variants

To reach a trade-off between PUF area, reliability and robustness to modeling attacks, the MPUF, along with its two variants, were proposed [4]. The primary goal was to reach robustness to cryptanalysis and modeling attack comparable to XOR PUF, while being much more reliable than the XOR APUF. An $(n, k)$-MPUF employs an $2^k : 1$ multiplexer, with the $k$ select lines of the multiplexer being connected to the outputs of $k$ APUFs, and the data lines of the multiplexer being connected to $2^k$ APUFs, with all AUFs are input the same $n$-bit challenge. Fig. 4 shows the architecture of an $(n, k)$-MPUF. In the same paper, the authors also proposed two other variants, viz., cMPUF (robust against cryptanalysis) and rMPUF (robust against reliability-based modeling attack). The rMPUF uses $2^{k-1}$ APUFs for data selection using a multiplexer tree, while the cMPUF uses only $2^{k-1}$ data APUFs, with the other $2^{k-1}$ data inputs being complements of them. The authors concluded through extensive theoretical and experimental analysis that the $(64, 3)$-rMPUF to have comparable robustness to modeling attack as an 10-XOR APUF with 64-bit challenge, and its reliability is as high as the reliability of a 4-XOR APUF. The authors demonstrate that MPUF variants can also achieve statistical properties similar to LSPUF without, using any additional input network.

### E. Interpose PUF (IPUF)

The *Interpose PUF* (IPUF) [17] is the latest addition to the repertoire of robust PUFs. The idea is to interpose the response of an $x$-XOR APUF (with $n$ challenge bits) to an applied challenge, between two challenge bits of an $y$-XOR APUF (with $(n + 1)$ challenge bits, the remaining challenge bits being the same as the $x$-XOR PUF), to get an $(x, y)$-IPUF. Fig. 5 shows the structure of an $(x, y)$-IPUF, with $n$-bit challenge. The authors claimed through theoretical analysis and experimental results) that an $(x, y)$-IPUF design while having the same hardware overhead and comparable reliability as an $(x + y)$-XOR PUF, provides much higher robustness to modeling attacks. The authors claimed that using the middle bit of the second APUF as the interpose position, the IPUF is robust against classical machine learning based modeling attack, reliability based modeling attack, and cryptanalytic attacks. In particular, the authors demonstrate that the $(3, 3)$-IPUF is satisfactorily resistant to all known attacks.

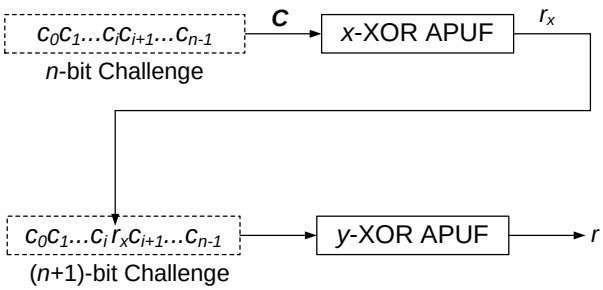


Fig. 5. Architecture of an $n$-bit $(x, y)$-IPUF [17]. Only the $n$-bit challenge $c$ and the 1-bit response $r$ are accessible as circuit ports.

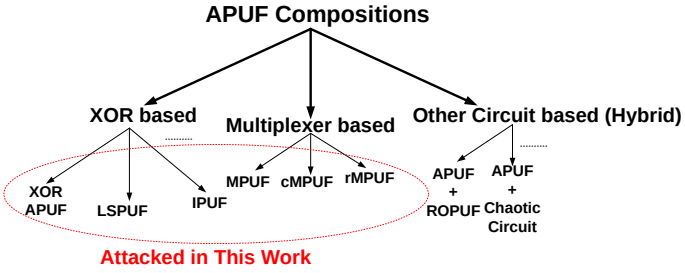


Fig. 6. Summary of the scope of this work.

### F. Our Observations about Robustness of PUFs to Modeling Attacks

As we demonstrate through our modeling attack results, many of the PUF variants claimed to be secure against modeling attack techniques tried previously, are actually vulnerable to deep learning based modeling attacks (details in Section IV. For example, we show that not only the $(3, 3)$-IPUF, but also the $(4, 4)$-IPUF is vulnerable to deep learning based modeling attack. Fig. 6 summarizes the scope of this paper.

We do not consider the feed-forward APUF (FF APUF) [1] in this work. As demonstrated in [10], although the FF APUF is largely resistant to SVM and LR based modeling attacks, even structurally complex FF APUFs with up to 8 feed-forward loops are quite susceptible to ES based machine learning attack. However, in spite of its severe vulnerability to a plethora of modeling attack techniques, we still examine the vulnerability of the APUF because it is the fundamental building block of the other PUFs considered in this work.

## III. Deep Learning Methodology for APUF Composition Modeling

Deep learning is a specialization of the *Artificial Neural Network* (ANN) concept in machine learning. In recent years, deep learning has demonstrated remarkable success in performing different complex tasks in the domain of artificial intelligence, especially for tasks like object classification and language translation [27], [28]. Deep Learning can be applied to supervised, semi-supervised and unsupervised learning problems. The most commonly used deep learning architectures are the *Deep Feedforward Neural Networks*, *Convolutional Neural Networks* (CNN), *Recurrent Neural Networks* (RNN), Autoencoders, *Restricted Boltzmann Machines*

(RBM), etc. Deep feedforward neural network architectures employ more than one hidden layer [28]. In [29], it has been shown that multilayer feedforward networks with as few as one hidden layer are indeed capable of universal approximation in a very precise and satisfactory sense. However, the number of parameters and required dataset size increases as the number of neurons increases in a single hidden layer. It is also possible to approximate complex functions with deep feedforward architecture with less number of neurons in each layer. This reduces the number of parameters to train and also requires relatively less number of examples [30]. In the proposed method, we use deep feedforward neural network architecture to model APUF compositions.

A neural network uses *activation functions* to learn complex mappings and patterns of the data. An activation function at a neural network node defines the relationship between its output value and input values. One of the most commonly used activation function is the *Sigmoid* activation function, which is a special case of the logistic function that is defined as: $\text{Sigmoid}(z) = \frac{1}{1+e^{-z}}$. It is bounded in the range [0, 1], has a positive derivative at each point, and its derivative can be computed very easily. However, it suffers from the *gradient vanishing* effect as its argument increases. However, post-2011, the most computationally efficient and popular activation function is the "Rectified Linear Unit" (ReLU), which is defined as: $ReLU(z) = \max(0, z)$. It allows the neural network to easily obtain sparse representations, and because of its piecewise linearity, there is no gradient vanishing effect [31]. The same work also showed that training of very deep neural networks is much faster if the hidden layers are composed of ReLU. Though ReLU is very powerful, it can only be used in the hidden layers and cannot be used in the output layer due to its form. ReLU is also the default activation function recommended for use with most feedforward neural networks [28]. We use deep feedforward networks with sigmoid activation function for the output layer and ReLU activation function for the hidden layers.

In neural networks, *backpropagation* is used to repeatedly adjust the weights of the connections in the network, so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. But the algorithm is sensitive to the initial weights assigned to the nodes of the network [32], which can drastically impact the convergence of the backpropagation algorithm. In our approach, we perform random initialization of the weights, based on the *uniform distribution* that is normalized based on the size of the input and output of the layers ("Glorot uniform" initialization in the *Keras* framework), as suggested in [33].

We used *Adam optimizer* [34] for updating the weights during backpropagation and *Binary Cross Entropy* as loss function. Binary cross entropy function is defined as :

$$L_{bce}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})\right] \quad (4)$$

where $y_i$ and $\hat{y_i}$ correspond to the original responses and predicted responses of the PUF, and $n$ is the number of responses considered. The generic deep feedforward architecture used for modeling APUF and its variants is shown in Fig. 7.

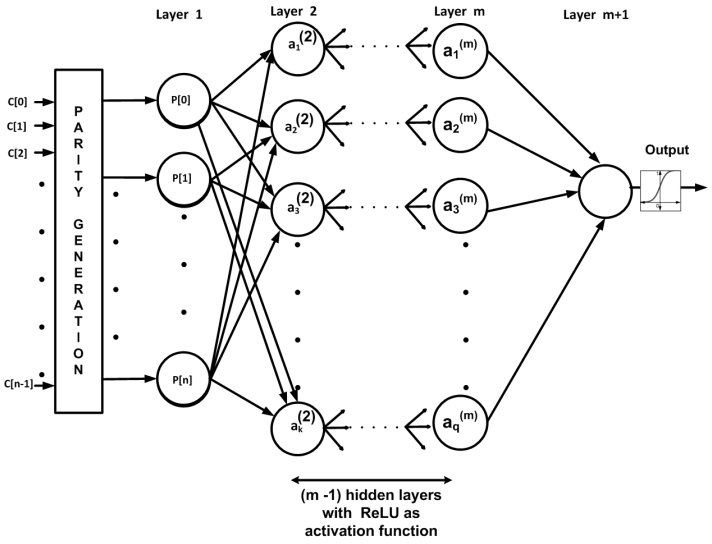


Fig. 7. Generic Deep FF Neural Network Architecture for modeling APUF Compositions.

Table I shows the different hyperparameters used in our deep feedforward neural networks.

A key advantage of deep feedforward neural networks here is "representation learning", which is a set of methods that allows a deep learning architecture to be input raw data and to automatically discover the representations needed for detection or classification [27]. For example, in our case the stack of hidden layers transforms the input parity vectors (calculated from the challenges) [1], into classifiable features automatically. The authors of [10] showed the learnability of APUF and $x$-XOR PUF ($x < 6$) using Logistic Regression. The deep feedforward architecture with sigmoid activation function at the output layer can be intuitively thought as the combination of two sequential steps: (a) transformation of the parity vectors to internally-generated features, and then (b) applying Logistic Regression on the learned features for classification.

## IV. Experimental Results

### A. Implementation and Modeling Setup

We currently do not have access to ASIC fabrication, and we found FPGA-based APUF implementations to exhibit very poor uniqueness, as also previously reported by multiple publications [4], [35], [36]. Hence, for our experiments, we collected CRPs from Matlab simulation models of the PUFs. Following [4], [10], we have considered that all stage delay parameters of each APUF are independent and identically distributed, and follow Normal distribution with mean $\mu = 0.1$ and $\sigma = 1$. to simulate imperfect reliability of actual PUF circuits, we have considered additive random noise which follows $\mathcal{N}(0, 0.01)$. For IPUF modeling, we used the Matlab-based PUF simulation model made available by the authors [37]. The modeling experiments were performed on 64-bit and 128-bit PUF variants. Random 128-bit and 64-bit challenges were generated (in sets of 1 million challenges), and applied to the PUF circuits as input. To calculate reliability, each PUF

TABLE I
IMPORTANT HYPERPARAMETER VALUES USED

| Hyper Parameters | Values Size (bits) |
|---|---|
| Kernel Initializer | *Glurot uniform* [33] |
| Learning Rate | *0.001* |
| Bias Initializer | *Zeros* |
| Optimizer | *Adam* [34] |
| Loss Function | *Binary cross entropy* |
| Hidden layer Activation function | *ReLU* [38] |
| Output layer Activation function | *Sigmoid* |

TABLE II
UNIFORMITY AND UNIQUENESS METRICS FOR SELECTED PUFS

| PUF | Challenge Size (bits) | Uniformity (%) (Mean, S.D.) | Uniqueness (%) (Mean, S.D.) |
|---|---|---|---|
| APUF | 64 | (51.04, 0.03) | (50.19, 0.04) |
| | 128 | (50.34, 0.02) | (50.35, 0.03) |
| 5-XOR APUF | 64 | (50.00, 0.00) | (49.98, 0.01) |
| | 128 | (50.03, 0.00) | (50.32, 0.01) |
| LSPUF $(m = 4, x = 4, k = 5)$ | 64 | (50.62,0.07) | (51.01,0.03) |
| | 128 | (49.97,0.00) | (49.14, 0.03) |
| MPUF $(k = 3)$ | 64 | (51.93, 0.07) | (54.20, 0.01) |
| | 128 | (52.61, 0.04) | (52.61, 0.02) |
| cMPUF $(k = 3)$ | 64 | (57.47, 0.07) | (50.92, 0.01) |
| | 128 | (57.47,0.07) | (50.92, 0.02) |
| rMPUF $(k = 3)$ | 64 | (50.05,0.03) | (46.28, 0.10) |
| | 128 | (50.40,0.02) | (47.11, 0.11) |
| (4,4)-IPUF | 64 | (50.50,0.00) | (50.00, 0.00) |
| | 128 | (50.11,0.00) | (49.98, 0.00) |

was characterized eleven times for the same challenge, and the response value obtained after majority voting the eleven responses were chosen to be the response of the PUF. To calculate uniqueness and uniformity, response sets for each PUF variant was considered for the same challenge sets. Table II shows the mean and standard deviation of the uniformity and uniqueness metrics for selected PUF variants. As we have seen in Section II-A, parity vectors linearly correlate with the delay difference of APUFs, and since we are attacking APUF compositions in this work, we chose to use parity vectors as input features. Our efforts of using the raw CRPs directly as input features was futile for all the PUF variants, as we got very poor accuracy results (between 55-60%), a phenomenon also reported in [21]. For each challenge, the parity vector [1] was calculated. These parity vectors, along with the corresponding responses, were used as input to the deep learning architecture, for training and testing purposes.

For each PUF, the generated CRP set was divided into two parts – the training set consisted of 80% of the total CRPs, while the test set consisted of the remaining 20%. The proposed deep learning modeling setup was implemented using Python 2.7 and the Keras 2.1.5 [39] framework, with TensorFlow [40] backend, and executed on a Linux workstation with 32 GB of main memory and a 3.3 GHz, 4-core Intel Xeon processor. All the experiments were conducted without us explicitly parallelizing the code across the cores, i.e., if multi-threaded execution happened, it was managed by the TensorFlow backend and was opaque to us. For the convenience of the readers and the research community, we have made the dataset and the modeling code available[2].

[2]https://github.com/Praneshss/Modeling_of_APUF_Compositions

TABLE III
MODELING ACCURACY RESULT FOR APUF

| Challenge Size (bits) | Training CRPs | Prediction Accuracy (%) | Training Time |
|---|---|---|---|
| 64 | 6,800 | 99.50 | 11.25 sec |
| 128 | 8,000 | 99.50 | 18.21 sec |

TABLE IV
MODELING ACCURACY RESULT FOR XOR APUF

| Challenge Size (bits) | No. of XORs | Training CRP Count | Prediction Accuracy (%) | Training Time |
|---|---|---|---|---|
| 64 | 2 | 32,000 | 99.30 | 56.36 sec |
| | 3 | 36,800 | 99.22 | 1 min 12 sec |
| | 4 | 41,200 | 98.60 | 2 min 10 sec |
| | 5 | 145,000 | 98.20 | 10 min 12 sec |
| | 6 | 680,000 | 97.68 | 20 min 52 sec |
| | 7 | 1,200,000 | – | – |
| 128 | 2 | 32,000 | 99.10 | 1 min 10 sec |
| | 3 | 37,600 | 98.90 | 2 min 5 sec |
| | 4 | 255,000 | 97.80 | 8 min 30 sec |
| | 5 | 655,000 | 97.87 | 29 min 21 sec |
| | 6 | 1,200,000 | – | – |

TABLE V
MODELING ACCURACY RESULT FOR LSPUF $(k = x + 1, m = x)$

| Challenge Size (bits) | No. of XOR $(\mathbf{x})$ | Training CRP Count | Prediction Accuracy (%) | Training Time |
|---|---|---|---|---|
| 64 | 3 | 80,000 | 98.50 | 3 min 20 sec |
| | 4 | 240,000 | 98.50 | 5 min 18 sec |
| | 5 | 320,000 | 97.23 | 18 min 35 sec |
| | 6 | 800,000 | 97.42 | 33 min 24 sec |
| | 7 | 1,200,000 | – | – |
| 128 | 3 | 80,000 | 97.82 | 3 min 10 sec |
| | 4 | 240,000 | 97.80 | 4 min 22 sec |
| | 5 | 1,200,000 | 96.22 | 3 hr 11 min |
| | 6 | 1,200,000 | – | – |

TABLE VI
MODELING ACCURACY RESULT FOR MPUF AND VARIANTS $(k = 3)$

| PUF Type | Challenge Size (bits) | Training CRP Count | Prediction Accuracy (%) | Training Time |
|---|---|---|---|---|
| MPUF | 64 | 111,000 | 98.10 | 2 min 5 sec |
| | 128 | 112,000 | 97.50 | 3 min 23 sec |
| cMPUF | 64 | 112,000 | 98.30 | 5 min 37 sec |
| | 128 | 112,000 | 97.50 | 4 min 5 sec |
| rMPUF | 64 | 80,000 | 98.20 | 5 min 3 sec |
| | 128 | 80,000 | 97.40 | 5 min 40 sec |

### *B. Modeling Accuracy Results*

In each case, we have reported the minimum number of CRPs that are required for successful modeling of the PUF variant. Tables XII–XVIII in the Appendix shows the number of hidden layers and average number of nodes per layer

TABLE VII
MODELING ACCURACY RESULT FOR MPUF AND VARIANTS ($k = 4$)

| PUF Type | Challenge Size (bits) | Training CRP Count | Prediction Accuracy (%) | Training Time |
|---|---|---|---|---|
| MPUF | 64 | 176,000 | 97.44 | 4 min 31 sec |
| | 128 | 184,000 | 96.49 | 16 min 10 sec |
| cMPUF | 64 | 112,000 | 97.36 | 5 min 07 sec |
| | 128 | 160,000 | 97.14 | 8 min 25 sec |
| rMPUF | 64 | 184,000 | 97.12 | 9 min 31 sec |
| | 128 | 264,000 | 96.23 | 20 min 16 sec |

TABLE VIII
MODELING ACCURACY RESULT FOR MPUF AND VARIANTS ($k = 5$)

| PUF Type | Challenge Size (bits) | Training CRP Count | Prediction Accuracy (%) | Training Time |
|---|---|---|---|---|
| MPUF | 64 | 256,000 | 97.02 | 14 min 13 sec |
| | 128 | 312,000 | 96.40 | 22 min 43 sec |
| cMPUF | 64 | 152,000 | 97.24 | 10 min 21 sec |
| | 128 | 215,000 | 96.36 | 10 min 13 sec |
| rMPUF | 64 | 320,000 | 96.54 | 15 min 23 sec |
| | 128 | 400,000 | 95.45 | 32 min 27 sec |

necessary for successful modeling. Also, we terminated the model building after 12 hours if model building had not converged even with 1.2 million CRPs as training data.

*1) Arbiter PUF:* Although modeling of APUF is not a challenging machine learning problem, for the sake of completeness we modeled 64-bit and 128-bit APUFs. For classification of APUFs, we found that the deep learning architectures necessary were particularly simple, e.g. only one hidden layer with two nodes was sufficient to model 64-bit APUF. Table III shows the results obtained on modeling APUFs. As expected, the modeling accuracy achieved is very high with relatively small training sets.

*2) XOR Arbiter PUF:* Table IV shows the results of modeled XOR APUFs. As seen in the table, 6-XOR APUFs for 64-bit challenges and 5-XOR APUFs for 128-bit challenges can be modeled quite accurately with far less computational effort than previously reported techniques [10], [26], although it requires more data. For example, in [10] it was reported that LR algorithm modeling for 5-XOR APUF with 128 bit challenge achieves 99% accuracy using 500,000 CRPs, but requires a training time of approximately sixteen hours [10]. The deep feed forward neural network achieves 97.87% accuracy using 655,000 CRPs in approximately half an hour.

*3) Lightweight Secure PUF:* We modeled individual output bits of LSPUF as in [10], and Table V reports the highest modeling accuracy for the output bits. In each case, there were $k = x + 1$ APUFs, and we set the number of output bits $m = x$, where each output bit depends on the XORing of $x$ APUF outputs, and the circular shift parameter $s = 0$. Again we find LSPUFs in which each output bit is up to 6-XOR for 64-bit challenges and 5-XOR for 128-bit challenges, can be modeled quite comfortably by deep learning. We consider a successful attack with only three hours of training to be a significant achievement, because previous effort of modeling 128-bit LSPUF (with 5-XOR outputs) required significantly larger training time of 267 days as reported in [10].

*4) Multiplexer PUF and Its Variants:* We found that the MPUF and its variants i.e., cMPUF and rMPUF (for the design parameter $k = 3, 4, 5$) can be modeled with relatively lesser number of CRPs than XOR-based APUF compositions. Tables VI–VIII shows results of modeled MPUF and its variants for both 64 bit and 128 bit challenges sizes. From the table it is evident that the complexity of the modeling attack increases as the value of $k$ increases.

*5) Interpose PUF:* We found the training data requirement of IPUFs is relatively higher than other XOR-based APUF compositions. The middle bit of the second APUF is used as interpose position for both 64-bit and 128-bit IPUFs, because this is supposed to provide the maximum robustness to modeling attack [17]. Table IX shows the modeling accuracy results for some IPUF variants. Note that we could successfully model up to (4,4)-IPUF for both 64-bit and 128-bit challenges. This is a major result, as mentioned earlier in Section II-F, because the authors of [17] found that previously proposed techniques cannot successfully attack even (3,3)-IPUF.

TABLE IX
MODELING ACCURACY RESULT FOR IPUF

| Challenge Size (bits) | IPUF Type $(x, y)$ | Training CRP Count | Prediction Accuracy (%) | Training Time |
|---|---|---|---|---|
| 64 | $(3, 3)$ | 240,000 | 98.30 | 6 min 29 sec |
| | $(4, 4)$ | 319,000 | 97.44 | 5 min 23 sec |
| | $(5, 5)$ | 1,200,000 | – | – |
| 128 | $(3, 3)$ | 288,000 | 97.47 | 10 min 21 sec |
| | $(4, 4)$ | 647,000 | 97.68 | 32 min 17 sec |
| | $(5, 5)$ | 1,200,000 | – | – |

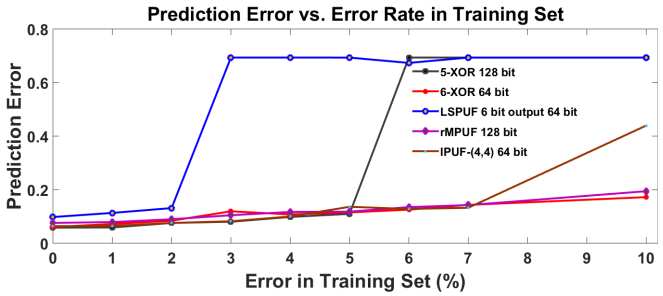


Fig. 8. Robustness results for various PUFs.

TABLE X
ROBUSTNESS ACCURACY RESULT FOR SELECTED PUFS

| PUF Type | Challenge Size (bits) | Training CRP Count | Pred. Acc. (No error) | Pred. Acc. (1% error) | Pred. Acc. (2% error) | Pred. Acc. (5% error) |
|---|---|---|---|---|---|---|
| 5-XOR | 128 | 655,000 | 97.87 | 97.79 | 97.42 | 96.93 |
| 6-XOR | 64 | 680,000 | 97.68 | 97.33 | 97.08 | 96.47 |
| LSPUF ($x = 6$) | 64 | 800,000 | 97.42 | 95.13 | 94.83 | 51.82 |
| rMPUF ($k = 3$) | 128 | 80,000 | 97.40 | 96.81 | 96.44 | 95.56 |
| (4,4)-IPUF | 64 | 320,000 | 97.44 | 97.33 | 97.30 | 95.66 |

TABLE XI
HIGHLIGHTS OF OUR EXPERIMENTAL RESULTS

| PUF Type | Challenge Size (bits) | Training CRP Count | Model. Acc. (%) | Training Time |
|---|---|---|---|---|
| LSPUF (5-XOR) | 128 | 1,200,000 | 96.22 | 3 hrs 11 min |
| (4,4)-IPUF | 128 | 647,000 | 97.68 | 32 min 17 sec |
| rMPUF ($k = 5$) | 128 | 400,000 | 95.45 | 32 min 27 sec |
| 5-XOR APUF | 128 | 655,000 | 97.87 | 29 min 21 sec |
| LSPUF (6-XOR) | 64 | 800,000 | 97.42 | 33 min 24 sec |
| 6-XOR APUF | 64 | 680,000 | 97.68 | 20 min 52 sec |
| (4,4)-IPUF | 64 | 320,000 | 97.44 | 5 min 23 sec |

### C. Robustness of Modeling Attack to Noisy Dataset

The impact of error in the training CRP set (possibly through corruption of the CRP database) was simulated by randomly flipping some percentage of response bits. The flipped training dataset is modeled and validated with the error-free test set. Table X shows variation in prediction accuracy, while Fig. 8 shows the robustness results for various selected PUFs, in terms of the change in prediction error, as the error rate in training data increases [10]. LSPUF shows early deviation with increasing error rates, size whereas 6-XOR and rMPUF (128 bit) shows higher and stable modeling accuracy even with 10% erroneous data. IPUF and 5-XOR (128-bit) show linear changes in the loss values till 5% and 7% respectively, after which the loss increases.

### D. Summary and Highlights of Results

The main observations from our results are as follows:

- For deep learning based modeling of different APUF compositions, the parity vectors seem to constitute a good input feature set, although we do not currently have a formal theory to justify this observation.
- In general, we found that modeling of 128-bit PUF versions are more challenging than the modeling of the 64-bit versions of the same PUF types, as the training data and the computation time required for the proposed modeling attack to succeed increases with the challenge size.
- Similar data requirement and training time trends were observed as the complexity of the PUF variants increase (e.g. modeling of 3-XOR APUF is more challenging than that of 2-XOR APUF).
- Increase in structural complexity of PUF is a stronger determining factor than increase in challenge size, regarding the modeling feasibility of the PUF variant.
- Deep learning modeling among notable machine learning attack resilient PUFs consistently outperforms previously proposed standalone (those which do not use any auxiliary information) machine learning based modeling techniques such as Logistic Regression in modeling PUFs, often achieving comparable modeling accuracy in much lesser computation time, although it also requires more training data in general.
- The input network of a LSPUF seems to make machine learning based modeling attack for individual output bits more difficult than XOR APUFs of the same complexity, an observation also reported in [10].
- In contrast to the conclusions reached in [4], robustness to modeling attacks of rMPUFs seems inferior to that of XOR APUFs.
- In contrast to the conclusions reached in [17], (3,3)-IPUF and (4,4)-IPUF can be modeled by deep learning.
- The deep learning based modeling technique demonstrates comparable robustness to error-inflicted training data as previously used machine learning techniques such as Logistic Regression.

Table XI presents the summary of the main results for the proposed attack technique on the different PUF variants considered, where we have included only the results which supersede other previously proposed (mainly [10]) standalone modeling attacks in effectiveness. **We foresee that through the application of more powerful computational infrastructure and more data resources (e.g. more CRPs), it would be feasible using deep learning, in near future to model PUFs using which evaded modeling in this work.**

We were unable to replicate the experiments described in the ANN based modeling attack technique on XOR APUFs reported in [25] for $x \geq 6$. The results presented in [25] are surprising because very large datasets have been operated upon, with relatively low-end computational resource, even inferior to the one used by us. For example, to model 128-bit 7-XOR APUF, they used a training dataset of chunk-size 14 GB on a laptop computer with 16 GB of main memory, and the training time was only 1.5 hours! The authors of [25] were contacted but did not respond to our requests for clarification at the time of submitting this manuscript.

## V. Conclusions

We have launched successful deep learning based modeling attack on APUF and its several compositions. The attack is computationally feasible for most practical PUF designs, and reasonably robust to input dataset noise. Our future works would be directed towards theoretical justification of the modeling accuracy trends we have observed, and applying the proposed technique to model other common PUF types such as the *Ring Oscillator PUF* [3] and its variants, and other novel APUF compositions, possibly combining APUF with other PUFs [41] or non-linear chaotic circuits [42].

## References


[1] D. Lim, "Extracting Secret Keys from Integrated Circuits," Master's thesis, Massachusetts Institute of Technology, U.S.A., 2004.

[2] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS'02, 2002, pp. 148–160.

[3] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Proceedings of the Design Automation Conference*, ser. DAC'07, 2007, pp. 9–14.

[4] D. P. Sahoo, D. Mukhopadhyay, R. S. Chakraborty, and P. H. Nguyen, "A Multiplexer-Based Arbiter PUF Composition with Enhanced Reliability and Security," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 403–417, March 2018.

[5] U. Ruhrmair and M. van Dijk, "PUFs in Security Protocols: Attack Models and Security Evaluations," in *Proceedings of the IEEE Symposium on Security and Privacy*, ser. SP'13, 2013, pp. 286–300.

[6] E. Simpson and P. Schaumont, "Offline Hardware/Software Authentication for Reconfigurable Platforms," in *Proceedings of Workshop on Cryptographic Harwdare and Systems*, ser. CHES'06. Springer Berlin Heidelberg, 2006, pp. 311–323.

[7] C. Helfmeier, C. Boit, D. Nedospasov, and J. P. Seifert, "Cloning Physically Unclonable Functions," in *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust*, ser. HOST'13, 2013, pp. 1–6.

[8] S. Tajik, H. Lohrke, F. Ganji, J.-P. Seifert, and C. Boit, "Laser Fault Attack on Physically Unclonable Functions," *Proceesings of the Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 85–96, 2015.

[9] S. Katzenbeisser, Ü. Kocabaş, V. Rožić, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon," in *Cryptographic Hardware and Embedded Systems – CHES 2012*, E. Prouff and P. Schaumont, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 283–301.

[10] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on Physical Unclonable Functions," in *Proceedings of the ACM Conference on Computer and Communications Security*, ser. CCS'10, 2010, pp. 237–249.

[11] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar, and W. P. Burleson, "Efficient Power and Timing Side Channels for Physical Unclonable Functions," in *Proceeding of Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES'14, 2014, pp. 476–492.

[12] F. Ganji, S. Tajik, and J.-P. Seifert, "Let Me Prove It to You: RO PUFs Are Provably Learnable," in *Proceeding of International Conference on Information Security and Cryptology*, ser. ICISC'15, S. Kwon and A. Yun, Eds. Springer International Publishing, 2015, pp. 345–358.

[13] F. Ganji, S. Tajik, F. Fäßler, and J.-P. Seifert, "Strong Machine Learning Attack Against PUFs with No Mathematical Model," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES'16, B. Gierlichs and A. Y. Poschmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 391–411.

[14] F. Ganji, S. Tajik, and J.-P. Seifert, "Why Attackers Win: On the Learnability of XOR Arbiter PUFs," in *Trust and Trustworthy Computing*, M. Conti, M. Schunter, and I. Askoxylakis, Eds. Springer International Publishing, 2015, pp. 22–39.

[15] ——, "PAC learning of Arbiter PUFs," *Journal of Cryptographic Engineering*, vol. 6, no. 3, pp. 249–258, Sep. 2016.

[16] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight Secure PUFs," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD'08, 2008, pp. 670–673.

[17] P. H. Nguyen, D. P. Sahoo, C. Jin, K. Mahmood, U. Rhrmair, and M. van Dijk, "The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks," https://eprint.iacr.org/2018/350, 2018, Accessed: May 2018.

[18] G. T. Becker, "The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES'15, T. Güneysu and H. Handschuh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 535–555.

[19] D. P. Sahoo, P. H. Nguyen, D. Mukhopadhyay, and R. S. Chakraborty, "A Case of Lightweight PUF Constructions: Cryptanalysis and Machine Learning Attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, pp. 1334–1343, 2015.

[20] P. H. Nguyen, D. P. Sahoo, R. S. Chakraborty, and D. Mukhopadhyay, "Security Analysis of Arbiter PUF and Its Lightweight Compositions Under Predictability Test," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 2, pp. 20:1–20:28, Dec. 2016.

[21] Y. Ikezaki and Y. Nozaki and M. Yoshikawa, "Deep learning attack for physical unclonable function," in *2016 IEEE 5th Global Conference on Consumer Electronics*, Oct 2016, pp. 1–2.

[22] R. Yashiro, T. Machida, M. Iwamoto, and K. Sakiyama, "Deep-learning-based security evaluation on authentication systems using arbiter puf and its variants," in *Advances in Information and Computer Security*, K. Ogawa and K. Yoshioka, Eds. Springer International Publishing, 2016, pp. 267–285.

[23] H. Shen, "Towards a Mathematical Understanding of the Difficulty in Learning with Feedforward Neural Networks," https://arxiv.org/pdf/1611.05827, Nov. 2017, Accessed: May 2018.

[24] K. Kawaguchi, "Deep Learning without Poor Local Minima," in *Proceedings of the Annual Conference on Advances in Neural Information Processing Systems*, ser. NIPS'16, 2016, pp. 586–594.

[25] Ahmad O. Aseeri and 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*, 2018, pp. 49–56.

[26] J. Tobisch and G. T. Becker, "On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation," in *Proceedings of International Workshop on Radio Frequency Identification: Security and Privacy Issues*, ser. RFIDSec'15, S. Mangard and P. Schaumont, Eds., 2015, pp. 17–31.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[29] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[30] G. Schwarz, "Estimating the Dimension of a Model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[31] X. Glorot, A. Bordes, and B. Y, "Deep Sparse Rectifier Neural Network," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. AISTATS'11, 2011, pp. 315–323.

[32] J. F. Kolen and J. B. Pollack, "Backpropagation is Sensitive to Initial Conditions," *Complex Systems*, vol. 4, no. 3, 1990.

[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, ser. AISTATS'10, 2010, pp. 249–256.

[34] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proceedings of the International Conference on Learning Representations*, ser. ICLR'15, 2015.

[35] Y. Hori, H. Kang, T. Katashita, A. Satoh, S. Kawamura, and K. Kobara, "Evaluation of Physical Unclonable Functions for 28-nm Process Field-Programmable Gate Arrays," *Journal of Information Processing*, vol. 22, no. 2, pp. 344–356, 2014.

[36] S. Morozov, A. Maiti, and P. Schaumont, "An Analysis of Delay Based PUF Implementations on FPGA," in *Reconfigurable Computing: Architectures, Tools and Applications*, P. Sirisuk, F. Morgan, T. El-Ghazawi, and H. Amano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 382–387.

[37] D. P. Sahoo, P. H. Nguyen, C. Jin, and K. Mahmood, "Defense/Attack PUF Library (DA PUF Library)," https://github.com/scluconn/DA_PUF_Library, 2018, Accessed: May 2018.

[38] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the International Conference on Machine Learning*, ser. ICML'10, 2010, pp. 807–814.

[39] F. Chollet *et al.*, "Keras: The Python Deep Learning library," https://keras.io, 2015.

[40] M. Abadi *et al.*, "TensorFlow: A System for Large-scale Machine Learning," in *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16, 2016, pp. 265–283.

[41] D. P. Sahoo, S. Saha, D. Mukhopadhyay, R. S. Chakraborty, and H. Kapoor, "Composite PUF: A new design paradigm for Physically Unclonable Functions on FPGA," in *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust*, ser. HOST'14, 2014, pp. 50–55.

[42] L. Chen, "A framework to enhance security of physically unclonable functions using chaotic circuits," *Physics Letters A*, vol. 382, no. 18, pp. 1195–1201, 2018.


## APPENDIX
## DEEP NEURAL NETWORK ARCHITECTURES

We present below the minimum number of hidden layers and the average number of nodes per layer required for successful modeling of the PUF variants, the accuracy results for which were presented in Section IV. **Note that these network architectures are not unique, and it is quite possible that other alternative deep neural network architectures can be derived which are equally or more effective in modeling the same PUF variants.**

TABLE XII
DEEP NEURAL NETWORK ARCHITECTURE FOR APUF

| Challenge Size (bits) | No. of Hidden Layers | No. of Nodes per Layer |
|---|---|---|
| 64 | 1 | 2 |
| 128 | 1 | 3 |

TABLE XIII
DEEP NEURAL NETWORK ARCHITECTURE FOR XOR APUF

| Challenge Size (bits) | No. of XORs | No. of Hidden Layers | Av. No. of Nodes per Hidden Layer |
|---|---|---|---|
| 64 | 2 | 2 | 6 |
| | 3 | 2 | 6 |
| | 4 | 2 | 15 |
| | 5 | 2 | 29 |
| | 6 | 5 | 230 |
| 128 | 2 | 2 | 5 |
| | 3 | 2 | 10 |
| | 4 | 3 | 53 |
| | 5 | 4 | 100 |

TABLE XIV
DEEP NEURAL NETWORK ARCHITECTURE FOR LSPUF ($k = 5, m = x$)

| Challenge Size (bits) | No. of XORs | No. of Hidden Layers | Av. No. of Nodes per Hidden Layer |
|---|---|---|---|
| 64 | 3 | 4 | 20 |
| | 4 | 4 | 30 |
| | 5 | 4 | 100 |
| | 6 | 5 | 400 |
| 128 | 3 | 3 | 20 |
| | 4 | 4 | 30 |
| | 5 | 4 | 400 |

TABLE XV
DEEP NEURAL NETWORK ARCHITECTURE FOR MPUF AND VARIANTS ($k = 3$)

| PUF Type | Challenge Size (bits) | No. of Hidden Layers | Av. No. of Nodes per Hidden Layer |
|---|---|---|---|
| MPUF | 64 | 2 | 30 |
| | 128 | 4 | 27 |
| cMPUF | 64 | 4 | 27 |
| | 128 | 4 | 27 |
| rMPUF | 64 | 3 | 30 |
| | 128 | 4 | 30 |

TABLE XVI
DEEP NEURAL NETWORK ARCHITECTURE FOR MPUF AND VARIANTS ($k = 4$)

| PUF Type | Challenge Size (bits) | No. of Hidden Layers | Av. No. of Nodes per Hidden Layer |
|---|---|---|---|
| MPUF | 64 | 3 | 50 |
| | 128 | 3 | 50 |
| cMPUF | 64 | 4 | 27 |
| | 128 | 4 | 40 |
| rMPUF | 64 | 3 | 50 |
| | 128 | 4 | 50 |

TABLE XVII
DEEP NEURAL NETWORK ARCHITECTURE FOR MPUF AND VARIANTS ($k = 5$)

| PUF Type | Challenge Size (bits) | No. of Hidden Layers | Av. No. of Nodes per Hidden Layer |
|---|---|---|---|
| MPUF | 64 | 3 | 55 |
| | 128 | 4 | 65 |
| cMPUF | 64 | 4 | 31 |
| | 128 | 3 | 60 |
| rMPUF | 64 | 3 | 70 |
| | 128 | 4 | 65 |

TABLE XVIII
DEEP NEURAL NETWORK ARCHITECTURE FOR IPUF

| Challenge Size (bits) | IPUF Type $(\mathbf{x}, \mathbf{y})$ | No. of Hidden Layers | Av. No. of Nodes per Hidden Layer |
|---|---|---|---|
| 64 | $(3, 3)$ | 3 | 50 |
| | $(4, 4)$ | 3 | 60 |
| 128 | $(3, 3)$ | 3 | 50 |
| | $(4, 4)$ | 3 | 60 |