Neural Network Modeling Attacks on Arbiter-PUF-Based Designs

Nils Wisiol*°, Bipana Thapaliya†° , Khalid T. Mursi‡, Jean-Pierre Seifert§ and Yu Zhuang¶
*§Technische Universität Berlin
†¶Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA
‡Department of Cybersecurity, College of Computer Science and Engineering, University of Jeddah, Jeddah, Saudi Arabia
Email: *nils.wisiol@tu-berlin.de, †bipana.thapaliya@ttu.edu, ‡kmursi@uj.edu.sa, §jean-pierre.seifert@tu-berlin.de, ¶yu.zhuang@ttu.edu

***Abstract*—By revisiting, improving, and extending recent neural-network based modeling attacks on XOR Arbiter PUFs from the literature, we show that XOR Arbiter PUFs, (XOR) Feed-Forward Arbiter PUFs, and Interpose PUFs can be attacked faster, up to larger security parameters, and with an order of magnitude fewer challenge-response pairs than previously known both in simulation and in silicon data. To support our claim, we discuss the differences and similarities of recently proposed modeling attacks and offer a fair comparison of the performance of these attacks by implementing all of them using the popular machine learning framework Keras and comparing their performance against the well-studied Logistic Regression attack. Our findings show that neural-network-based modeling attacks have the potential to outperform traditional modeling attacks on PUFs and must hence become part of the standard toolbox for PUF security analysis; the code and discussion in this paper can serve as a basis for the extension of our results to PUF designs beyond the scope of this work.**

***Index Terms*—Physical Unclonable Function, Strong PUFs, Machine Learning, Modeling Attacks, Arbiter PUF.**

I. Introduction

In all cryptographic applications deployed today, what distinguishes the legitimate user from an adversary is the knowledge of the secret keys, which are found anywhere cryptography is used, including in computers of microscopic scale embedded in digital door keys, credit cards, and passports. As cryptography became more ubiquitous, gaining access to the secret key itself became an important attack strategy, as the revealed secret key causes the security guarantees of the employed scheme to collapse.

To mitigate such attacks, a branch of research on *Physically Unclonable Functions (PUFs)* emerged [20], where the difference of the legitimate user and adversary is not defined by knowledge, but by possession of a physical object. The physical object, called PUF token, is assumed to exhibit highly individual physical behavior when prompted with a physical stimulus such as an electrical signal or laser beam. Further, it is assumed to be *physically unclonable*, meaning that it is inherently impossible to produce two identical tokens. The envisioned cryptography shall be based on this unique response behavior of each individual unclonable token instead of classical secrets stored in memory.

°Nils Wisiol and Bipana Thapaliya contributed equally to this work.

While such PUF-based, secret-free cryptography is by definition immune against attacks that recover any secret key, adversaries may be able to study the individual behavior of a PUF token and extrapolate it using a mathematical model, in which case it is impossible for a remotely connected party to tell the original PUF token and mathematical model apart. If such *modeling attacks* succeed, the security of most applications would collapse, just like in the case of a leaked secret key. (Note that, on the other hand, inability to model a PUF successfully does not by itself guarantee its security.)

An important tool to launch modeling attacks is *machine learning*, a highly parameterized approach to create predictions from observed data by using specialized algorithms. In the past two decades, machine learning has emerged as the tool of choice for the security analysis of PUFs [11], [21], where an attacker attempts to create a model of a PUF token which, if successful, can be used to predict PUF responses with high accuracy. Studies on modeling attacks hence represent an essential part of research on PUFs and secret-free cryptography. With the growing popularity in both science and industry as well as the rapid development of machine learning software frameworks such as Tensorflow/Keras or Torch, we expect this to remain the case for the foreseeable future.

Traditionally, most machine-learning-based modeling attacks on strong PUFs have been based on the optimization of the parameters of a physically motivated model of the strong PUF. More recently, this approach was complemented by a modeling attack methodology based on general models not derived from physical insights [34], [18], [3], [17], [23], [32] and models that are extensions of a physical model [23]. Generic models in this sense are no longer restricted to a certain class of functions derived from physical inspiration, but are able to model any Boolean function. Thus, such attacks do not require exact modeling of the PUF under attack and allow for rapid testing of PUF design ideas. While also generic models benefit from feature selection based on physical insights, using general neural networks such as the multilayer perceptron as an analysis tool for strong PUF design has found some adoption, and is in particularly helpful in the analysis of designs composed of several Arbiter PUFs.

For example, in a recent modeling attack on the Interpose PUF by Wisiol et al. [32], the authors used such general

models to demonstrate that also slight variations of the PUF under attack are not promising candidates for a secure strong PUF. In another modeling attack study, Santikellur et al. [22] use a multilayer perceptron approach to study the security of the MPUF, cMPUF, rMPUF, Lightweight Secure PUF, XOR Arbiter PUF, and Interpose PUF. The authors of the SCA-PUF [37] show that their proposed design is more resilient than an XOR Arbiter PUF with respect to attacks based on neural networks. In the security analysis of their novel PUF design based on a subthreshold voltage divider array, Venkatesh et al. [29] provide a failed modeling attack using a multilayer perceptron as evidence for the security of their proposal.

The quick advancement and rising popularity in modeling attacks based on neural networks raise questions regarding the significance of the results obtained using such general-purpose models. To estimate a PUF design's security level, it is particularly important to know, first, how the data and time complexity of general neural-network-based attacks relate to more specialized physical-model-based attacks, second, how the chances of success of these two types of attacks relate, and, third, how the various hyperparameters of such attacks need to be configured.

In this work, we answer the above questions with respect to the Arbiter PUF and its variants, a family of electrical PUF designs that is commonly used in PUF research as a baseline for comparison or as a building block for novel PUF designs. With the contributions of this work, other researchers are enabled to conduct similar analyses for other PUF designs. In more detail, our contributions are:

1) We show that XOR Arbiter PUFs can be attacked faster, up to larger security parameters, and with an order of magnitude fewer challenge-response pairs than previously known by using generic neural networks, even if the implementation gives reliable responses. We thereby show that neural networks are an essential tool in PUF security analysis and refute statements to the contrary of Nguyen et al. [19] made at CHES 2019.
2) We show that our results reduce the data complexity of the Splitting Attack on the Interpose PUF and enable attacks on the Feed-Forward Arbiter PUF, thus demonstrating that the improved performance of neural-network-based attacks has implications beyond the security of the XOR Arbiter PUF itself.
3) We replicate three neural-network-based attacks and the physical-model-based Logistic Regression attack on XOR Arbiter PUF from the literature and provide an exhaustive and fair comparison of their performances. We overview and summarize all four attacks, discussing differences and similarities, and justify design choices.
4) We refute a recently claimed very low data complexity of XOR Arbiter PUF modeling attacks [17].
5) We provide a comprehensive, unified, and easy-to-use Python implementation of all attacks in this work under an open-source license at https://github.com/nils-wisiol/pypuf/tree/2021-mlp, integrated in the pypuf framework [31].

This paper is organized as follows. In the next section, we give an overview on work that relates to modeling attacks on XOR Arbiter PUFs and modeling attacks using neural networks. In Sec. IV, we discuss and evaluate the Logistic Regression attack as a baseline for comparison of attacks in this work. In Sec. V, VI, and VII, we replicate and extend neural-network-based attacks from the literature and discuss design choices as well as attack performance, with a comparison of the XOR Arbiter PUF attacks given in Sec. VIII. We apply our findings to the Interpose PUF in VII-D, to XOR Arbiter PUF silicon data in Sec. VII-E, and to the XOR Feed-Forward Arbiter PUF in Sec. IX. We draw conclusions and discuss future work in Sec. X.

II. Related Work

The security analysis of PUFs is not limited to modeling attacks based on machine learning, and not limited to the attacker model used in this work.

For XOR Arbiter PUF, the state-of-the-art attack operates on information about the reliability with respect to a given challenge, i.e., on information about the probability that the PUF token will produce the same response when given the same challenge [5]. Given this information, the data complexity of the modeling of XOR Arbiter PUFs is drastically reduced, even compared to the results of neural-network-based attacks in this work. Recently, Tobisch et al. demonstrated that this attack methodology can be extended to other PUF designs and launched an attack on the Interpose PUF [26]. Chatterjee et al. [7] used a reliability-based attack to show that the security level of the S-PUF is drastically reduced within this attacker model.

In contrast to these works, the related work below consider a weaker attacker model in which only the response information is considered.

Highly specialized attacks based on machine learning are also an essential part of the security analysis of PUFs, as demonstrated by Delvaux [8] in a collection of five attacks on PUFs with accompanying lightweight obfuscation logic. This approach contrasts the generic models used in this work.

Another branch of security analysis of PUFs uses the Probably Approximately Correct (PAC) framework [28], which is concerned with classes of functions for which an asymptotically polynomial-time learning algorithm exists that will yield a model of prescribed accuracy with prescribed success rate. In certain cases, this can be done without using a physically inspired model [10], which relates to the application of generic neural networks in this work. Recently, a semi-automated approach for analysis was proposed [6].

III. Preliminaries

A. Additive Delay Model

An n-bit k-XOR Arbiter PUF [11], [24], as displayed in Fig. 1, can be modeled using a Boolean function $f : \{-1, 1\}^n \to \{-1, 1\}$ which is parameterized by real values $W \in \mathbb{R}^{k \times n}$ and $b \in \mathbb{R}^k$. The response to a challenge $c \in \{-1, 1\}^n$ is given by

$$f(c) = \operatorname{sgn} \prod_{l=1}^{k} \left(\langle W_l, x \rangle + b_l \right).$$

In this setting, the *feature vector* $x \in \mathbb{R}^n$ is a function of the given challenge c defined by $x_i = \prod_{j=i}^{n} c_j$. The parameterization W, called *weights* in this work, represents the physical intrinsics of a particular PUF token. This model is commonly referred to as the *additive delay model*. A physical motivation, derived from the intrinsic *delay* values of an Arbiter PUF circuit, can be found in full version [33].

In the case of Arbiter PUFs, i.e., $k = 1$, the additive delay model can be understood as a hyperplane in n dimensions, dividing the edges of the Boolean cube $\{-1, 1\}^n$ (and, by extension, $\mathbb{R}^n$) into two regions labeled by the -1 and 1 responses of f. The boundary between the two regions is a linear, or in case of nonzero bias, affine subspace of $\mathbb{R}^n$. In this sense, the model can be understood as *linear*. In an XOR Arbiter PUF, the decision boundary in $\mathbb{R}^n$ becomes more complex, hence adding XOR operations increases the *nonlinearity* of the model.

This linearity of the Arbiter PUF model is also the motivation of using the feature map $x_i = \prod_{j=i}^{n} c_j$. Without it, the decision boundary cannot easily be represented as a hyperplane in $\mathbb{R}^n$. This, however, is a prerequisite for the successful application of the Logistic Regression attack [21] (cf. Sec. IV).

Most modeling attacks in this work are based on simulated challenge-response data, with the application of our attack to silicon data given in Sec. VII-E. The reason for this is twofold. First, security analysis with a focus on the PUF design should be done on ideal data, i.e., without regard to the properties of a specific implementation. This ideal data should follow the *expected* properties of implementations. In the case of the Arbiter PUF, as pointed out by Tobisch and Becker [27], this means that different PUF tokens may have varying ML resistance. Any implementation's security should be studied separately in case the security analysis of the ideal primitive is promising. Second, while FPGA implementations of Arbiter PUFs suffer from known weaknesses, ASIC implementations are expensive and are well known to behave very closely as predicted by the additive delay model [9], [25].

For the modeling of noise, we rely on the model by Delvaux et al. [9]. In this model, for each evaluation of the PUF, a Gaussian noise value with zero mean is added to the delay difference of the two delay lines in the Arbiter PUF. That is, the Arbiter PUF response is modeled as

$$f(c) = \operatorname{sgn}\left(\langle W, x\rangle + b + N\right),$$

where N is chosen from a Gaussian distribution with zero mean and defined variance, $W \in \mathbb{R}^n$ and $b \in \mathbb{R}$ model the Arbiter PUF physical properties, and $x \in \{-1, 1\}^n$ is the feature vector corresponding to the given challenge c. This model extends to k-XOR Arbiter PUFs by drawing k independent noise values.

To measure how the noise influences the behavior of the PUF, we define the *reliability* of the PUF as the expected value over the challenge space that the PUF will return the same response when given the challenge twice in independent evaluations,

$$\text{reliability}(f) = \mathop{\mathrm{E}}_{c}\left[f^{(1)}(c) = f^{(2)}(c)\right],$$

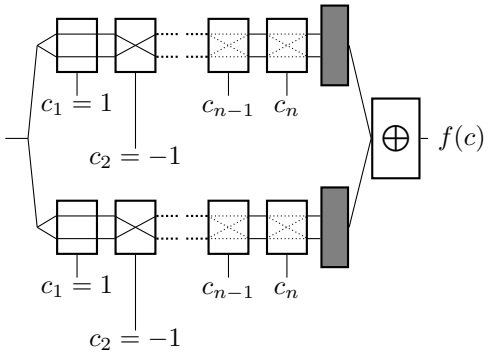


Fig. 1. Schematic representation of a 2-XOR Arbiter PUF with n-bit challenge c and final response $f(c)$.

where $f^{(1)}(c), f^{(2)}(c)$ are independent evaluations of the PUF token.

Given this definition, we can adjust the variance of the noise such that the reliability of our simulation reaches a certain value.

B. Attacker Model

The standard attacker model for PUFs is based on the promise of increased hardware security of PUF tokens, compared to the traditional approach of cryptographic hardware extended by secure key storage. Hence, in the standard PUF attacker model, the adversary gets physical access to the PUF token for some limited amount of time, allowing them to collect a large number of challenge response pairs. (During that time, also hardware attacks are allowed, but not studied in this work.)

Given this attacker model, XOR Arbiter PUFs and Interpose PUFs can be attacked using information that is exposed by the per-challenge reliability, which the attacker can recover by evaluating a challenge multiple times [5], [26].

In this work, however, we consider the weaker attacker model used in the works that we replicate, where the *passive attacker* only gets access to a large collection of uniformly random challenges and the corresponding responses of the PUF. While this is, in general, an unjustified restriction of the PUF attacker model, it is equivalent to a situation where the reliability information of the PUF under attack is either not available or not useful. This is the case in some PUF-based authentication protocols [16], [35], [36], and may be the case for some novel PUF designs, as a weaker justifiable attacker model is desirable for the PUF designer. Thus, the passive attacker model serves an appropriate basis for our argument that neural-network-based modeling attacks should become part of the standard security analysis of PUFs. Nevertheless, due to the restricted attacker model, we emphasize that a failed modeling attack in this attacker model cannot by itself serve as evidence for security.

C. Methodology

To provide a fair comparison of the properties of four different machine learning modeling attacks on XOR Arbiter

PUFs, in this work, we reimplemented these attacks using the popular Keras framework and ran all experiments on the same CPU. Also, our novel attacks on the Interpose PUF and Feed-Forward Arbiter PUFs are using this framework. As the attacks run, compared to previous works, relatively fast, we focus our attention on the comparison of the data complexity, i.e., how much training data is required to obtain good modeling predictions.

In this work, we do not focus on the exact accuracy metrics of prediction quality for several reasons. First, the attacks that we study do not yield intermediate results when correctly parameterized, i.e., the attacks end either with accuracy 50% or close to 100%. Additionally, we observed that high-accuracy results can be further improved by letting the training continue for a few more epochs. Second, to impersonate a PUF token, no extremely high accuracy is needed, and any prediction accuracy significantly better than 50% should be considered a security weakness of the PUF design [8]. Hence, instead of prediction accuracy, we focus on the attack *success rate*, i.e., the proportion of independently run attacks on independent PUF simulations that yielded prediction accuracy greater than 90%. Given that virtually all successful attacks yielded accuracy 95% or better and virtually all unsuccessful attacks yielded accuracy 55% or below, the success rate is very insensitive to the exact choice of this threshold. Nevertheless, for the sake of completeness, the prediction accuracy of obtained models is given as the average over several attacks on different PUF tokens.

To ensure that all of our results are reproducible in detail, we seed all involved pseudorandom number generators used for generating PUF token simulations, initializations of machine learning models, etc. with defined values.

IV. Baseline: Improved Logistic Regression Attack

In this work, we compare three neural network modeling attacks against the commonly used *Logistic Regression* (LR) attack by Rührmair et al. [21] on XOR Arbiter PUFs. The LR attack has become an important tool for the security analysis of delay-based PUFs and is used in a large number of different works, including in the recent proposal of the Interpose PUF [19] and an attack on it [32]. In this work, we use our Keras implementation of the LR attack with a few modifications detailed below as the baseline for comparison of XOR Arbiter PUF attacks.

The LR attack is based on a physical model of the Arbiter PUF, which can be represented as a hyperplane in n-dimensional space by converting a given challenge into an appropriately chosen feature vector. A physical motivation for the feature vector used in this and previous works can be found in the full version of this paper [33]; if the Arbiter PUF uses any logic to transform the challenge before applying it to the delay paths, this needs to be taken into consideration as well. This Arbiter PUF model was extended to the XOR Arbiter PUF using the observation that the XOR operation can be written in a differentiable way by representing the Arbiter PUF response values by -1 and 1 and the XOR operation with the real product of these values.

A large-scale study of Tobisch and Becker [27] determined training set sizes for the LR attack that yield optimal results, i.e., have the lowest training times, and minimal training set sizes, for which the attack was observed to work at least once, which we confirmed and will use for comparison. To provide for a fair comparison with the neural network attacks, we reimplemented the LR attack in the Keras machine learning framework. Differences in run time hence may not only be caused by the usage of different CPUs, but also by optimization differences in the implementations. Nevertheless, we obtain the same number of required CRPs, which indicates that our implementation behaves similar to the one of Tobisch and Becker.

To increase the training performance of the original LR algorithm, we modified some details. First, to reduce the number of epochs required for training, we introduced the usage of mini batches, where the network is updated with the gradient not only after evaluating the complete training data, but several times in each epoch. This allows for convergence using fewer epochs, and thus (except in corner cases), for shorter training time, but one must be careful to not choose the batch size too small. Too small batch sizes can lead to noisy gradient values, which will in turn perform unhelpful updates on the network.

Second, we use Adam optimizer [12] instead of the originally used resilient backpropagation, as the latter works poorly together with the use of mini batches.

Third, we apply the tanh activation function to each of the k delay values computed by the respective arbiter chains, i.e. we change the model function from

$$f_{\mathrm{LR}}(c) = \tanh\left(\prod_{l=1}^{k}\left(\langle W_l, x\rangle + b_l\right)\right) \quad \text{to} \quad f'_{\mathrm{LR}}(c) = \tanh\left(\prod_{l=1}^{k}\tanh\left(\langle W_l, x\rangle + b_l\right)\right). \tag{1}$$

This change was motivated by the observation that in the traditional LR network, a single arbiter chain can have large influence on the absolute value of the final output. However, in the electrical circuit, no analogue to the absolute value exists. Instead, XOR Arbiter PUF model weights can be scaled using positive scalars without affecting the computed function. We speculate that different influences can hamper the training process, as weight updates during backpropagation may be applied predominantly to influential arbiter chains. Applying the tanh function ensures a more equalized influence of all arbiter chains on the final output, as factor values are bounded by an absolute value of 1.

By the introduction of the additional tanh activation function, our attack does not fulfill the definition of logistic regression anymore[1]. In a slight abuse of terminology, we will refer to this version as the *improved LR attack*. A sketch of the network structure used in the attack is displayed in the full version of this paper [33].

[1]One can argue that already the LR attack on XOR Arbiter PUF hardly fits the textbook definition of a logistic regression.

TABLE I
EMPIRICAL RESULTS ON LEARNING SIMULATED XOR ARBITER PUFS OBTAINED USING OUR KERAS-BASED IMPLEMENTATION OF THE LR ATTACK. REFERENCE VALUES OF TOBISCH AND BECKER [27] USE UP TO 16 CORES. (* RESULT OBTAINED USING A DIFFERENT NUMBER OF CRPS.) WE USED 4 THREADS FOR ATTACKS WITH $k \leq 8$; 40 THREADS FOR $k \geq 9$. THE ATTACKS FOR $k = 9$ AND $k = 10$ USED 132GIB AND 197GIB OF MEMORY, RESPECTIVELY.

k	CRPs	success rate	duration (threads)	mean success accuracy	[27]
4	30k	10/10	1 min	95.5%	1 min
5	260k	10/10	4 min	96.9%	1 min
6	2M	20/20	1 min	97.6%	1 min
7	20M	10/10	3 min	98.5%	55 min
8	150M	10/10	28 min	96.4%	391 min
9	500M	7/10	14 min	96.8%	*2266 min
10	1B	6/10	41 min	95.7%	-

Using our Keras-based implementation together with these improvements, we could increase the performance of the LR algorithm (with respect to wall clock time, i.e., elapsed real-time from beginning to end of the model training procedure), which is summarized in Tab. I. We found that the largest proportion of the performance gain is due to Keras, which allows for optimized and highly parallel computing, and to a smaller extent due to our improvements. In a comparison of attack performance on 64-bit 5-XOR Arbiter PUFs, we found that our implementation of the original LR attack achieved a success rate of 16/20, while the improved version yielded 18/20 successful, with all other parameters being equal. We did not study in more detail the performance of the two versions depending on the choice of mini batch size, number of CRPs, learning rate, etc, as the neural-network attacks presented in this work outperform the LR attack by an order of magnitude in data complexity. We thus performed such analysis on the neural-network attack (Fig. 2).

Similar to the attacks based on neural networks shown below, performance of the improved LR attack crucially depends on the choice of hyperparameters, in particular on a good combination of learning rate and batch size. The number of required epochs is also heavily influenced by any early stopping logic, which may depend on the validation accuracy or loss. We thus expect that the wall-clock performance and the number of required epochs can be further reduced, e.g., by using a systematic approach to find optimal hyperparameters. On the other hand, we expect that for the LR attack, the data complexity cannot significantly be reduced by hyperparameter tuning, as our results are in-line with previous research on the LR attack [21], [27], [30], [32].

Our numbers confirm once more [21], [32] that the LR attack requires a number of CRPs in the training set that grows exponentially with the number of employed XORs in the target XOR Arbiter PUF. In Fig 4, we show the required training set size. Based on a fitted function $k \mapsto \alpha \cdot e^k$ using the least squares method, we predict the number of required CRPs for $k = 10$ is 1.3 billion, for $k = 11$ is 3.6 billion, and for $k = 12$ is 10^{12}.

V. TENSOR REGRESSION NETWORK ATTACK BY SANTIKELLUR ET AL.

To reduce the computational effort for modeling attacks on XOR Arbiter PUFs, Santikellur et al. [23] proposed to use an efficient CP-Decomposition Tensor Regression Network (ECP-TRN), which is parameterized by an integer rank R. To model an k-XOR n-bit Arbiter PUF, the proposed model computes the function

$$f(x) = \operatorname{sgn}\left[\sum_{i=1}^{R}\left(\alpha_i \cdot \prod_{l=1}^{k}\langle w_{i,l}, x\rangle + b_{i,l}\right)\right],$$

where $w_{i,l} \in \mathbb{R}^n$ and $b_{i,l} \in \mathbb{R}$. A drawing of the network structure is shown in the full version of this paper [33]. Due to the highly parallel structure of the network, the approach may benefit from performance improvements during training. The parameters to be trained are α_i, $i \in \{1, \ldots, R\}$ and $w_{i,l}, b_{i,l}$, $l \in \{1, \ldots, k\}, i \in \{1, \ldots, R\}$. Hence, there are $nkR+kR+R$ trainable parameters.

Given this network structure, the network can be understood as an approach that trains R XOR Arbiter PUF models in parallel. The final response of the model is then computed as the weighted sum of the R model outputs, then the sign of the response is returned. After the training completes, the network is filled with R sets of XOR Arbiter PUF weights, which raises the question how the *individual* prediction accuracy differs from the *overall* prediction accuracy reported by Santikellur et al.

For our experiments operating on simulations of 4, 5, and 6-XOR Arbiter PUFs with 64-bit challenge lengths, we found that in all successfully trained networks, exactly one of the R trained models showed high correlation with the simulation weights, whereas the other $R - 1$ had no correlation. This finding was confirmed by the prediction accuracy: $R-1$ of the models in the successfully trained network had an individual prediction accuracy of around 50%, whereas exactly one had high prediction accuracy. Using the single model allowed for even higher prediction accuracy than using the fully trained network, as the noise of the $R - 1$ low-correlation models is removed. Considering that there is hardly any improvement in loss when more than one model predicts the correct answer, we can conclude that the R-rank model of the ECP-TRN does not provide benefits over the 1-rank model.

As the final response of the ECP-TRN network is computed as the weighed sum of the R individual model responses, the individual models influence each others training process through the backpropagation algorithm. To examine if this interdependency during training provides an advantage to the modeling attack, we run many *attack attempts* on a single PUF under attack, i.e., we restart the training process with different initializations of the model, while keeping the PUF simulation and CRP set constant. In the case of many attack attempts, the training of each attempt is independent of the training process of the other attempts, above-mentioned interdependency is removed. This allows us to compare the performance metrics of rank R ECP-TRN attacks using a single attack attempt versus rank 1 ECP-TRN using R attack attempts. The results are displayed in Tab. II.

TABLE II
COMPARING SINGLE-ATTEMPT ATTACKS USING RANK R ECP-TRN VERSUS R-ATTEMPT ATTACKS USING RANK 1 ECP-TRN ON 64-BIT XOR ARBITER PUFS. OUR RESULTS INDICATE THAT THE TRAINING OF THE ECP-TRN DOES NOT BENEFIT FROM INTERACTION OF THE MODELS; BUT GIVES SOME INDICATION TO THE CONTRARY. COMPARED TO THE FIGURES OF SANTIKELLUR ET AL. [23], FOR $k = 5$ AND $k = 6$, WE TRIPLED AND DOUBLED THE NUMBER OF CRPS, RESPECTIVELY, TO OBTAIN ANY SUCCESSFUL RESULTS.

k	CRPs	R	attempts per run	total runs	run success rate	mean attempt success acc.
4	40k	1	5	10	100%	95.2%
4	40k	5	1	10	100%	94.6%
5	320k	1	10	10	90%	94.9%
5	320k	10	1	10	90%	94.9%
6	800k	1	10	10	80%	95.1%
6	800k	10	1	10	80%	95.0%
7	800k	1	100	5	20%	95.4%
7	800k	1000	1	4	0%	—

In none of the cases that we studied, we found that the training of a rank R ECP-TRN did not surpass the success rate of just running R independent learning attempts using a rank 1 ECP-TRN.

Unfortunately, we were not able to replicate the ECP-TRN results of Santikellur et al. [23] exactly as published in the original paper. While the 64-bit 4-XOR case could be replicated, our experiments for 5-XOR and 6-XOR required significantly more CRPs for reliable convergence than originally claimed. For 7-XOR and larger, we failed to achieve any success using the proposed high-rank model. A discussion with the original authors also could not improve our results. We suspect that the reason for the larger requirement of CRPs is either caused by the different behavior of Keras internals compared to the original Tensorflow v1 implementation, or, considered more likely, by some differences in CRP generation. It may also play a role that by training R models in parallel, the original authors effectively report the maximum accuracy among (a large number of) R learning attempts, while our work reports data complexities for success rates mostly close to 100%. We further discuss this in Sec. VIII and X.

VI. MULTILAYER PERCEPTRON ATTACK BY ASEERI ET AL.

After an attack on 3-XOR 64-bit Arbiter PUFs by Yashiro et al. [34] using a neural network with autoencoders and an attack by Alkatheriri et al. [2] on Feed-Forward Arbiter PUFs using a multilayer perceptron, Aseeri et al. [3] were the first ones to attack XOR Arbiter PUFs with more than four arbiter chains using neural networks.

While much of their work focused on the fact that their version of the modeling attack can be run on a regular laptop computer, i.e., on a machine without GPU, but with limited memory and just using a single core of a consumer CPU, their attack also achieves a significant reduction in both time and data complexity, compared to the then state-of-the-art LR attack by Tobisch and Becker [27].

Some attempts to replicate the work of Aseeri et al. failed, [22], likely because at the time, the source code of the attack was not publicly available yet. Consequently, the attack was not sufficiently considered in the security analysis of the Interpose PUF [19]. We take this as evidence that the publication of code that is the basis for claimed attack results should be strongly encouraged by the community.

TABLE III
EXTENDED RESULTS ON LEARNING SIMULATED XOR ARBITER PUFS OBTAINED USING OUR KERAS-BASED IMPLEMENTATION OF THE MULTILAYER PERCEPTRON ATTACK BY ASEERI ET AL. [3]. MEMORY USED BY OUR IMPLEMENTATION IS PROPORTIONAL TO THE NUMBER OF CRPS, WITH THE LARGEST ATTACK REQUIRING 50GIB FOR 100M CRPS WHEN $n = 128$. *TO ALLOW FOR COMPARISON WITH THE ORIGINAL FIGURES, WE COMPUTED AN APPROXIMATION OF THE DURATION USING A SINGLE CORE. THE PERFORMANCE LOSS IS CAUSED BY THE LOWER SINGLE-CORE PERFORMANCE OF OUR CPUS (INTEL XEON E5-2630 V4) COMPARED TO ASEERI ET AL.'S (INTEL CORE I7). ALL OF OUR EXPERIMENTS USE UP TO 40 THREADS; ASEERI ET AL. USED ONLY ONE.

n	k	CRPs	success rate	duration (40 threads)	success accuracy	duration [3] (1 thread)
64	4	400k	10/10	<1 min	96.8%	<1 min
64	5	400k	10/10	<1 min	96.7%	<1 min
64	6	2M	9/10	<1 min	97.3%	7 min
64	7	5M	9/10	<1 min	97.3%	12 min
64	8	30M	10/10	3 min	98.1%	23 min
64	9	80M	9/10	86 min	98.2%	-
128	4	400k	10/10	<1 min	96.8%	1 min
128	5	3M	10/10	<1 min	97.1%	5 min
128	6	20M	10/10	<1 min	98.0%	19 min
128	7	40M	10/10	5 min	97.9%	90 min
128	8	100M	1/10	45 min	98.6%	-

The original implementation of this attack was done using scikit learn. As part of our comparison of neural network attacks, in this work, we reimplemented the network used by Aseeri et al. using the Keras machine learning framework and were able to replicate all of their results. An overview of our replicated results can be found in Tab. III, including an extension to the 64-bit 9-XOR and 128-bit 8-XOR cases. While the original figures strictly use single-core performance on a consumer CPU, we used up to 40 cores in parallel. To allow for comparison, we include an estimation of the single-core performance of our implementation by multiplying the measured wall clock time with the maximal number of threads our experiment allowed. This overestimates the time required by our attack, especially for cases where multi-threading allows only for little speedup, i.e., for small training sets.

Aseeri et al. did not include arguments for the specific hyperparameter settings they used in their attack. We include a discussion of the multilayer perceptron hyperparameters in Sec. VII. A comparing overview can be found in Tab. V; a drawing of the network can be found in the full version of the paper [33].

VII. MULTILAYER PERCEPTRON ATTACK BY MURSI ET AL.

A. Neural Network

In follow-up work to Aseeri et al. [3] and Santikellur et al. [22] (not to be confused with the ECP-TRN model), Mursi et al. [17] presented an enhancement of the multilayer perceptron

XOR Arbiter PUF modeling attack, claiming to reduce the data and time complexity of XOR Arbiter PUF modeling attack by several orders of magnitude. We refute their empirical results, but show that their attack still requires fewer CRPs than other response-based XOR Arbiter PUF modeling attacks. Consequently, we are able to demonstrate that XOR Arbiter PUFs can be attacked up to higher security parameters than previously known, posing a challenge to implementors who need to keep the noise low enough to allow for the fabrication of XOR Arbiter PUFs with such large security parameters.

To attack a n-bit k-XOR Arbiter PUF, Mursi et al. propose to use a neural network that consists of three fully connected hidden layers of sizes $2^{k-1}, 2^k, 2^{k-1}$. We depict such a network in the full version of this paper [33]. By its design, this model uses fewer trainable parameters than the MLP-approach by Aseeri et al. and the high-rank approach by Santikellur et al., which can benefit training. Nevertheless, it uses orders of magnitude more parameters than the traditional LR attack. For example, in the attack of a 64-bit 9-XOR Arbiter PUF, LR uses 585 trainable parameters, while the MLP attack in this section uses 66,560.

Mursi et al. also use the tanh activation function for the hidden layers, compared to the usage of ReLU by Aseeri et al. This is beneficial to the learning process as the inputs to the next layer are normalized, i.e. have zero mean [14]. While Santikellur et al. [22] argue that tanh suffers from the vanishing gradient problem, the successful use of tanh in the MLP attack can be explained by the relatively shallow three-layer structure of the neural network, which makes the vanishing gradient problem unlikely to appear [17]. A detailed comparison of hyperparameters as used in the different attacks showed in this paper can be found in Tab. V.

B. Replication and Results

To make the various neural-network-based attacks comparable, we reimplemented the attack by Mursi et al. using the Keras machine learning framework and found that their results could not be replicated. The difference in attack performance of our implementation and the original could be traced back to a bug in the CRP generator used by Mursi et al., which was based on a simulation of the delays. For each PUF instance, $4n$ delays were supposed to be drawn independently from a Gaussian distribution with mean 300 and variance 40; given a challenge, the delay difference can then be computed and converted into the PUF response. Due to the bug, which we discovered during our independent replication of the results using pypuf [31], about 20% of the randomly drawn delays were inadvertently set to zero. This was difficult to notice from the CRP data, as the bias was hardly influenced and the MLP-based attack does not recover the simulation delays or weights, but a neural network that is hard to be interpreted.

To study the attack by Mursi et al., we use our reimplementation of the neural network attack and the pypuf CRP generator [31] used throughout this work and in a recent LR-based attack on the Interpose PUF [32]. The CRP generation is, for performance reasons, based on the equivalent approach of using weights instead of delays. We found that while the attack performance reported by Mursi et al. significantly benefited from the faulty CRP generation, the results obtained on valid CRPs still improve on the LR attack in terms of data complexity by an order of magnitude, with increasing advantage for an increasing number of XORs, cf. Fig. 4. We also observed an improvement in run time. Detailed results are reported in Tab. IV.

For challenge lengths 128 and 256, we found that the data complexity grows fast with the number of XORs. Nevertheless, for 128 bit challenges, it remains below the figures that Tobisch and Becker [27] reported for the LR attack; for 256 bit challenges we could not find numbers in the literature to compare to. However, the LR attack is known to have polynomially increasing data complexity in the challenge length [32]. The steeply increasing required number of CRPs of the MLP attack could be caused by an inherent effect of the XOR Arbiter PUF structure, or by a mismatch of hyperparameters or neural network structure on the model. Considering everything, we conclude that there is no evidence that increasing the challenge length will be an effective defense against modeling attacks and let this question open to be studied in case sufficiently large XOR Arbiter PUFs can be built.

In light of the reduced data complexity, we come to the conclusion that a model with far more trainable parameters is able to outperform a model with fewer model parameters, which falsifies the claim by Nguyen et al. that the LR attack is the best performing among the XOR Arbiter PUF attacks [19].

By the detailed comparison of the attacks run by Mursi et al. with our results, we can identify the faulty CRP generation as the sole cause for the different data complexity of the attack. Hence, as a byproduct of our replication of the attack, we find that a relatively small proportion of zero-valued delays in the XOR Arbiter PUF can lead to a large loss of data complexity for the modeling attack. With this in mind, implementors of PUFs should treat any significant deviation from simulation-based attack results on real-world data with additional scrutiny on the validity of their implementation. In future security analyses of PUFs, the detailed validity of the simulation in use needs to be established, otherwise the analysis could over- or underestimate the PUF's security. Such validation is especially challenging when using generic models such as the MLP for modeling attacks. However, in case of the Arbiter PUF, several independent results confirming the validity of the additive delay model exist [11], [25], [9].

C. Hyperparameter Optimization

As a technical note, we found it difficult to configure the hyperparameters of the attack by Mursi et al. While the processing of the training data in mini batches provides benefits regarding the run time, and thus the development of the attack, it also requires to adjust the learning rate appropriately. In Fig. 2 we report the success rate of MLP-based attacks on 8-XOR 64-bit for a large variety of different configurations of learning rate and batch size, showing that

only an appropriate combination of those two hyperparameters will yield a successful attack. We speculate that on the one hand, for high learning rates on small batches, the gradient direction is too noisy to yield a meaningful update to the model, and on the other hand, that for low learning rates on large batches the learning process runs into the maximum number of epochs before a convergence could be achieved.

D. Noise Resilience and Application to the Splitting Attack

During our experiments, we found the attack to work in the presence of noise without significant changes in data complexity when compared to the noise-free case. This is surprising, as the MLP attack, in contrast to the LR attack, is not restricted to functions of a certain class. Nevertheless, the training converges to the desired XOR Arbiter PUF model and approaches the maximum predictive power. We report detailed results on the noise resilience in Tab. IV.

We also report that the MLP-based attack can be used as a drop-in replacement of the LR attack in the Splitting Attack on the Interpose PUF [19], [32]. We found that, similar to the results we obtained on XOR Arbiter PUFs, the data complexity of the Splitting Attack can be significantly reduced. Experiments using our MLP-based implementation of the splitting attack showed that a 64-bit $(1, 7)$-Interpose PUF can be modeled with 6 million CRPs in minutes, compared to 20 million CPRs and 20 hours required by the original implementation. A 64-bit $(1, 8)$-Interpose PUF was modeled using our MLP attack in less than one hour using 18 million CRPs (i.e., with data complexity less than previously reported for the $(1, 7)$ case).

In the splitting attack on the Interpose PUF, essentially two XOR Arbiter PUFs are attacked. First, the attack on the lower layer uses all available CRP data, then the attack on the upper layer of the Interpose PUF can only use about one half of the available CRP data. Compared to the XOR Arbiter PUF, this results in effectively doubling the data complexity of the splitting attack for Interpose PUFs of size (x, x), and no increase in data complexity for designs (x, y) where $x < y$. We extrapolate our attack results on the Interpose PUF (above) and the XOR Arbiter PUF (Tab. IV) and conclude that the MLP attack is able to attack 64-bit (1,11) Interpose PUFs using 325M CRPs (Wisiol et al. [32] originally used 750M to attack the (1,9) version), and 650M CRPs to attack a 64-bit (11,11)-Interpose PUF. While this has two orders of magnitude larger data complexity than the attacks by Tobisch et al. [26], the MLP attack provides better convergence rate and faster computation time.

As the splitting attack relies on a training set in which one feature is chosen at random [32], these results demonstrate that the MLP-based attack is (to some extend) resilient against feature noise, which makes it suitable in a variety of different attack scenarios, such as when XOR Arbiter PUF attacks are used as a building block in modeling attacks on PUFs that are composed of XOR Arbiter PUFs. Our results on noisy XOR Arbiter PUFs demonstrate that the MLP attack is (to some extend) resilient against label noise.

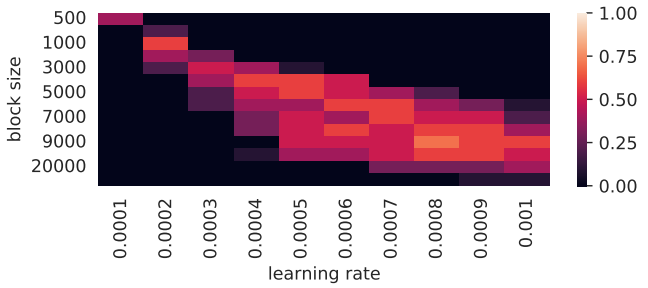


Fig. 2. Success rate for training an 8-XOR Arbiter PUF, 6M CRPs, with the attack by Mursi et al. [17], with varying learning rates and block sizes. For each combination, 10 learning attempts were run.

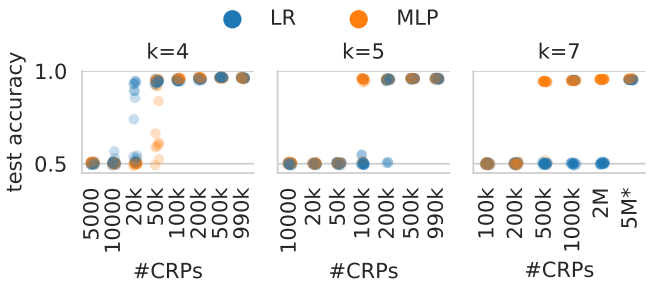
TABLE IV
EMPIRICAL RESULTS ON LEARNING SIMULATED n-BIT k-XOR ARBITER PUFS OBTAINED USING OUR IMPROVED IMPLEMENTATION OF THE NEURAL NETWORK ATTACK BY MURSI ET AL. [17]. THE LEARNING WAS CONFIGURED TO STOP AT VALIDATION ACCURACY 95%, THE VARIANCE OF ADDED NOISE WAS CONFIGURED SUCH THAT THE SIMULATION ACHIEVES THE GIVEN RELIABILITY VALUE. SUCCESS RATES ARE 10/10 FOR MOST ROWS SHOWN, OTHERWISE AT LEAST 7/10. ("—" INDICATES NO SUCCESSFUL RUN) RUN TIMES ARE GIVEN IN MINUTES.

k	CRPs	time [min] (threads)	success acc.	memory	[17] CRPs	[17] time
fully reliable responses, $n = 64$						
4	150k	<1 (40)	97.0%	1 GiB		
5	200k	<1 (20)	97.3%	3 GiB	42k	<1
6	2M	<1 (40)	97.5%	2 GiB	255k	2
7	4M	<1 (40)	97.5%	2 GiB	680k	1
8	6M	13 (4)	95.5%		1.7M	5
9	45M	16 (40)	98.1%	14 GiB	4.2M	9
10	119M	291 (40)	97.9%	41 GiB		
11	325M	1898 (40)	98.1%	104 GiB		
fully reliable responses, $n = 128$						
4	1M	<1 (40)	97.3%	1 GiB		
5	1M	<1 (20)	97.4%	3 GiB		
6	10M	<1 (20)	98.1%	5 GiB		
7	30M	2 (20)	98.2%	20 GiB		
fully reliable responses, $n = 256$						
4	6M	1 (40)	97.7%	6 GiB		
5	10M	2 (20)	97.8%	10 GiB		
6	30M	— (20)	—	30 GiB		
85% reliability, $n = 64$						
4	180k	<1 (4)	90.9%	1 GiB		
5	150k	<1 (4)	91.2%	1 GiB		
6	2M	<1 (4)	91.8%	1 GiB		
7	4M	3 (4)	91.6%	2 GiB		

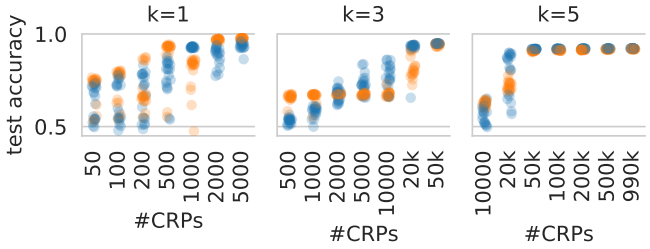
E. Application to Real-World Data

To confirm the lower data complexity of the MLP attack for realistic challenge-response and noise data, we compare the improved LR attack and MLP attack on the (XOR) Arbiter PUF data sets provided by Mursi et al. [17] and Aghaie and Moradi [1].

The data of Mursi et al. contains one instance of each an 64-bit k-XOR Arbiter PUF for $k \in \{4, \ldots, 9\}$, queried on the same set of 1 million (for $k \in \{4, 5, 6\}$) and 5 million challenges (for $k \in \{7, 8, 9\}$). This data set does not provide repeated measurements, hence no statement about the reliability can be made. Our statistical analysis of the



(a) Data collected by Mursi et al. [17].



(b) Data collected by Aghaie et al. [1].

Fig. 3. Comparison of improved LR attack (Sec. IV) and our MLP attack (Sec. VII) on FPGA data. These results are in line with our results on simulated data: the MLP attack shows lower data complexity for XOR Arbiter PUFs with a large number of XORs k. (*Training set size was actually 4,990,000; 10,000 CRPs were used for testing.)

responses confirmed the expectation of decreasing bias with an increase in the number of XORs; the 4-XOR Arbiter PUF has an average response of 0.03, the 9-XOR Arbiter PUF an average response of 10^{-5} (with its Boolean responses given as -1 and 1).

The data of Aghaie and Moradi [1] contains a 64-bit (1,5)-Interpose PUF queried on 1 million uniformly random challenges, each including the responses of all 6 individual Arbiter PUFs. The data also contains 11 repeated measurements of all challenges, which we only used to compute the reliability of the PUFs (all Arbiter PUFs had 99.7% or better). For the attacks, we just used the first of the 11 measurements, discarding the other 10. We confirmed the quality of the data by testing that all six involved Arbiter PUFs individually can be modeled using the delay model with high accuracy (all 98%). To use the Interpose PUF data in the context of XOR Arbiter PUF attacks, we discard the (64-bit) top layer and compose the five given 65-bit Arbiter PUFs of the Interpose bottom layer to 65-bit k-XOR Arbiter PUFs for $k \in \{1, \ldots, 5\}$. The data shows that the involved Arbiter PUFs are heavily biased, with 3 out of 5 have an average response of 0.3 or higher, computed from Boolean responses given as -1 and 1.

We ran the LR and MLP attack with varying number of CRPs on the XOR Arbiter PUFs obtained from the data to compare the data complexity of the attacks. The detailed results are displayed in Fig. 3. All attacks ran within a few seconds to few minutes each.

The results on the data of Mursi et al. show that for k-XOR Arbiter PUFs with $k = 5$ and $k = 7$, the data complexity of the

TABLE V
PARAMETER COMPARISON OF NEURAL-NETWORK-BASED MODELING ATTACKS ON k-XOR ARBITER PUFS.

	Santikellur et al. [23]	Aseeri et al. [3]	Mursi et al. [17]
Method	TRN	MLP	MLP
Architecture	many delay mod.	$(2^k, 2^k, 2^k)$	$(2^{k-1}, 2^k, 2^{k-1})$
Hid. lay. activ.	—	ReLU	tanh
Optimizer	Adam	Adam	Adam
Loss function	BCE	BCE	BCE
Learning rate	(multiple)	10^{-3}	adaptive
Initializer	Glorot Normal	Glorot Unif.	Gaussian

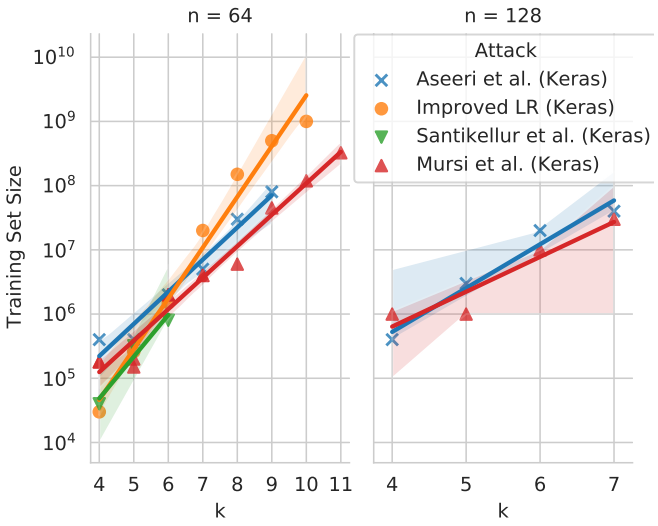


Fig. 4. Data complexity of our attacks on n-bit Arbiter PUF with k individual arbiter chains as determined as the lowest number of CRPs (among exponentially increasing options) that yielded at least 50% successful attacks, where successful means prediction accuracy of 90% or better. Our implementation of the attack by Mursi et al. outperforms the improved LR attack by an order magnitude with regard to data complexity.

MLP is much lower than that of the improved LR attack. (Note that the x-axis is scaled approximately logarithmically.) On the other hand, the results on the data of Aghaie and Moradi show that the MLP attack is either unsuited or ill-parameterized for k-XOR Arbiter PUF with smaller numbers for k and provides no advantage or improved LR in data complexity, like we observed for results we obtained on simulated CRPs (Fig. 4). As there is already a large variety of attacks available in this setting, we did not explore further if the neural-network attacks can be optimized for these sizes. We conclude that our experiments on real-world data confirm our findings obtained with simulated data.

VIII. COMPARISON OF XOR ARBITER PUF ATTACKS

Of the three studied neural-network-based modeling attacks and the improved LR attack used as a baseline for comparison, we found that the claims by Mursi et al. [17] could be traced back to an error in CRP generation, which leaves the work by Santikellur et al. [23] to claim, to the best of our knowledge, the lowest data complexity of XOR Arbiter PUFs in the

literature. However, we were unable to replicate these attacks for the cases of 7-XOR and larger.

Among the attacks successfully replicated in this work, we found the attack by Mursi et al. [17] to have, despite a relatively high number of trainable parameters, the lowest data complexity. Being an enhancement of neural network attacks presented by Aseeri et al. [3] and Santikellur et al. [22] (not to be confused with the ECP-TRN), we attribute the advantage in data complexity to the choice of network size and hyperparameters, concluding that a further reduction of complexity may well be possible for more carefully optimized settings. Optimizing further for the XOR Arbiter PUF, however, will not yield novel results, as the XOR Arbiter PUF must be considered broken under reliability attacks [5]; optimization effort should be invested in neural-network attacks on other PUF designs.

Comparing the data complexity of the MLP-attack by Mursi et al. to the results obtained with our improved version of the LR attack, we find that MLP only has advantages in data complexity for XOR Arbiter PUFs with more than four arbiter chains, but not for smaller designs. Next to the above-mentioned steep increase of data complexity for larger challenge lengths of the MLP-attack (cf. Sec. VII), we read this as evidence that the neural network structure is not optimal with respect to arbitrary values of challenge length and number of XORs.

Comparing the complexity of the MLP-attack by Mursi et al. with the MLP-attack by Aseeri et al., we find that the main differences of the attacks are the network shape and the choice of activation function in the hidden layers. While Mursi et al. demonstrated that attacks on XOR Arbiter PUFs are possible on consumer laptops, our results show that the time and data complexity of attacks can be further reduced. As discussed above, using the tanh activation function instead of ReLU provides outputs (inputs to the next layer) that are on average zero, which is beneficial to the training process [14].

For a detailed comparison of the four modeling attacks, graphs of the respectively used network structure are shown in the full version of this paper [33]. Chosen hyperparameters are shown in Tab. V, an overview over the data complexities is provided in Fig. 4.

We do not include a detailed comparison of run times, as most of the attacks presented in this work run in minutes, which makes a valid comparison difficult. Furthermore, using the Keras-based implementation, our attacks can be run in a variety of different settings, i.e., on CPUs and GPUs, with and without multi-threading, which will lead to different run times. In any event, none of the attack times reported in this work are prohibitively long for an attacker.

IX. Multilayer Perceptron Attack on Feed-Forward Arbiter PUFs

To illustrate our argument that MLP attack should become part of the standard security analysis of Strong PUF designs, we apply our attacks to Feed-Forward Arbiter PUFs and variants thereof.

Feed-Forward Arbiter PUFs were first introduced by Gassend et al. [11] and Lee et al. [15] and are based on the idea of introducing non-linearity to the response behavior (as discussed in Sec. III-A) by adding more arbiter elements that pick up the signal on the delay lines before they reach the last stage. These arbiter elements produce additional challenge bits which will be used in later stages. As such, the Feed-Forward Arbiter PUF can be thought of as a predecessor of the Interpose PUF, using the same Arbiter PUF instead of an additional layer of PUFs to produce additional, attacker-unknown, challenge bits. We will refer to the additional arbiter elements and challenge bits as feed-forward *loops*. An extension of the Feed-Forward Arbiter PUF is the (homogeneous) XOR Feed-Forward Arbiter PUF [4], where the result bit is – similar to the XOR Arbiter PUF – determined by a number of k individual Feed-Forward Arbiter PUFs with identical loop placements.

The Feed-Forward Arbiter PUF shows much stronger modeling attack resistance than the XOR Arbiter PUF. Rührmair et al. [21] attack only the Feed-Forward Arbiter PUF whose loops are arranged in regular patterns. Kumar and Burleson [13] demonstrated attacks on silicon data of Feed-Forward Arbiter PUFs with up to 8 loops, using an attack based on evolution strategies. They report data complexity much lower than our attack allows, but did not report results on XOR Feed-Forward Arbiter PUFs. Alkatheiri and Zhuang [2] use an MLP-based approach for learning, however their attack shows declining accuracy for an increase in the number of loops. Furthermore, the attack also requires the loop pattern to be known to the attacker, a condition that will not easily hold since different Feed-Forward Arbiter PUFs can have different loop patterns [2, personal communication]. While this deficiency could be alleviated by either an (computationally expensive) brute-force search or via a physical attack on the circuit, they do not report attack results on XOR Feed-Forward Arbiter PUFs.

In this work, we attack n-bit Feed-Forward Arbiter PUFs and XOR Feed-Forward Arbiter PUFs with up to 10 loops. To run the attack, we use an MLP of fitted size. In contrast to the MLP network we used to attack the XOR Arbiter PUF, this network is composed of four hidden layers with $n, {}^{n}/_{2}, {}^{n}/_{2}, n$ neurons, respectively. In our experiments, increasing the number of hidden layers beyond this resulted in a decrease in predictive power, increase of training time, or both. The network shape and hyperparameters are chosen independently of the number of loops and their positioning, hence no knowledge of this information is required to run the attack. As argued for the XOR Arbiter PUF (see Sec. VII), we opted to choose the tanh activation function over the common choice of ReLU. The attack network is displayed in the full version of this paper [33]. We target both simulated PUFs, with the simulation based of an appropriate adaptation of the additive delay model (Sec. III-A), and PUFs implemented on FPGAs.

We implemented 64-stage Feed-Forward Arbiter PUFs on three Artix®-7 FPGAs using the Xilinx Vivado design suite that consists of an editable MicroBlaze CPU. VHSIC Hardware Description Language (VHDL) was used to build the Feed-Forward Arbiter PUFs designs. The placement of each Feed-Forward Arbiter PUF on the chip was carried out horizontally on the chips using the Tool Command Language

TABLE VI
RESULTS OF ATTACKING 64-STAGE AND 128-STAGE SIMULATED FF PUFS USING OUR KERAS BASED IMPLEMENTATION OF THE MULTILAYER PERCEPTION COMPARED WITH THE MULTILAYER PERCEPTION ATTACK BY ALKATHEIRI ET. AL. [2]. THEIR METHOD ASSUMES THE LOOP PATTERN IS KNOWN TO THE ATTACKER, THE PROPOSED METHOD HAS NO SUCH ASSUMPTION. EACH SHOWN ROW HAS A SUCCESS RATE OF 10 OUT OF 10 TRIALS.

n	no. of loops	CRPs	duration	acc.	Alkatheiri et. al.[2] CRPs	duration	acc.
64	4	135k	6 min	95%	200k	1.5 min	89%
64	6	315k	10 min	93%	200k	6.2 min	87%
64	8	540k	21 min	92%	—	—	—
64	10	630k	26 min	93%	—	—	—
128	1	36k	2 min	96%	20k	1 min	95%
128	3	180k	8 min	93%	100k	1 min	92%
128	5	405k	17 min	93%	200k	1.4 min	87%
128	7	900k	35 min	92%	—	—	—
128	9	1.2M	45 min	91%	—	—	—

TABLE VII
RESULTS OF ATTACKING 64-STAGE SIMULATED k-XOR FEED-FORWARD ARBITER PUFS. RESULTS FOR ODD k ARE OMITTED FOR BREVITY.

loops	k	success CRPs	success rate	duration	accuracy	memory
1	2	120k	10/10	0.5 min	98%	<1 GiB
	4	540k	10/10	2.8 min	98%	2.6 GiB
	6	900k	10/10	7 min	98%	4.3 GiB
	8	6M	7/10	6 hrs	96%	10 GiB
2	2	180k	10/10	5 min	97%	1 GiB
	4	720k	10/10	32 min	98%	3.5 GiB
	6	2.7M	9/10	1.4 hrs	97%	9.8 GiB
	8	9.5M	9/10	22 hrs	96%	18.8 GiB
3	2	360k	10/10	8 min	97%	1.8 GiB
	4	900k	10/10	2.3 hrs	96%	3.3 GiB
	6	3.15M	10/10	9.2 hrs	94%	8.1 GiB
	8	13.5M	6/10	40 hrs	91%	21 GiB
4	2	900k	10/10	1.4 hrs	97%	3.7 GiB
	4	2.7M	9/10	8.3 hrs	96%	6.9 GiB
	6	9M	6/10	26 hrs	94%	13 GiB
	8	18M	4/10	46 hrs	92%	27 GiB
5	2	1.8M	10/10	4.9 hrs	95%	6.4 GiB
	4	7.2M	7/10	19 hrs	93%	14 GiB
	6	11.7M	5/10	32 hrs	91%	22 GiB
	8	18M	4/10	24 hrs	90%	30 GiB

(TCL). AXI Universal Asynchronous Receiver Transmitter (UART), with baud rate of 230kbits/second, was used to speed up the CRPs transformation between the Tera Term terminal and the FPGAs. The Xilinx SDK was utilized to program the input/output workflow of the CRPs generation from the chips. The implementation was done on three FPGA chips. We generated five million CRPs out of each implemented silicon PUF. The CRPs were generated at an ambient temperature of around 22°C, and core voltage set to 1.0V using the built-in chips resistor.

Our results show that our MLP-based method is able to model 64-stage Feed-Forward Arbiter PUFs with 10 loops and 64-stage 8-XOR Feed-Forward Arbiter PUFs when using 5 homogeneous loops per Arbiter PUF. Note that the challenge length of the PUF is reduced by the number of loops inserted, however our results also indicate no fundamental change in attack performance even when the challenge length is doubled to approx. 128 bit. The detailed results on simulated data is shown in Tab. VI for Feed-Forward Arbiter PUFs and in Tab. VII for XOR Feed-Forward Arbiter PUFs.

In our experiments, increasing the depth of the neural network and/or using different activation functions degraded performance of the attack compared to the results shown in this work.

Our experiments with real-world data largely confirm the attacks on 64-bit Feed-Forward Arbiter PUFs, independently of the choice of loop pattern, with the attack on 10 loops requiring a lightly larger amount of 770,000 CPRs. As the XOR operation of the XOR Feed-Forward Arbiter PUF is done in Boolean logic and noise-free, we expect that our results in simulation also transfer to real-world data on XOR Feed-Forward Arbiter PUFs. A table showing the results is omitted for brevity.

A further extension of the k-XOR Feed-Forward Arbiter PUF can be made by introducing *heterogeneous* loops, i.e. by using individual loop placements on the k involved Feed-Forward Arbiter PUFs. We report that the network of our MLP attack is unable to attack heterogeneous XOR Feed-Forward Arbiter PUFs even for moderate parameter settings involving just one loop per Arbiter PUF and $k = 3$. Modeling accuracy was saturated at around 60%. We hence believe that the neural network structure will need major modifications, such as more layers, more nodes, or a structure different from fully connected, to allow successful training for this extension and encourage further research in this direction.

Our results demonstrate that the MLP attack can model variants of the Feed-Forward Arbiter PUFs that previously were out of reach for modeling attacks. This underlines our argument that MLP attacks should be part of the security analysis of future PUF designs and further reduces the security level of the Feed-Forward Arbiter PUF family.

X. CONCLUSION

In this work, we compared three neural-network-based XOR Arbiter PUF modeling attacks [23], [3], [17] against a baseline given by an improved version of the Logistic Regression modeling attack. While we could not replicate experiments that claim extremely low data complexity of attacks, our results prove that even XOR Arbiter PUFs with perfectly reliable responses can be attacked faster, with far fewer challenge-response pairs than previously known, and consequently up to much security parameters. As increasing the security parameter of XOR Arbiter PUFs is technologically challenging, our results cast further doubt that XOR Arbiter PUFs of sufficient security level can be fabricated. Given the large hyperparameter space of neural-network-based attacks, it is well possible that with more optimization, the attack performance can be further improved.

Providing the implementations of all four attacks studied in this work in a common framework and tested on a common platform, we are able to provide a fair comparison of their performances and confirm that previously reported better performances are not just due to better frameworks or faster

CPUs, but are an intrinsic property of the strategy employed by Mursi et al.'s multilayer perceptron-based modeling attack [17]. We further demonstrated that neural network modeling attacks can be used to attack (XOR) Feed-Forward Arbiter PUFs for the security parameters typically studied in the literature. By applying the attack to silicon data of XOR Arbiter PUFs and Feed-Forward Arbiter PUFs, we confirm that these advantages carry over to implementations of these PUFs.

The MLP's advantage in data complexity and modeling power lets us conclude that neural-network-based attacks should become part of the standard toolkit for PUF security analysis, as on the one hand, they allow for rapid testing for PUF design ideas [32], and on the other hand, as we showed in this work, they may be able to provide lower attack time or data complexity. However, we explicitly do not advocate for neural-network-based attacks to *replace* the study of PUF design by specialized attacks inspired from physical models. In the case of XOR Arbiter PUFs, the physically inspired model still has important relevance to the MLP-attack, as it provides the features on which the modeling is based. Hence, attempting the MLP attack without any knowledge of the physical model would be to no avail, as the neural-network-based attacks studied in this work fail when operating on challenge bits, except for toy-sized XOR Arbiter PUFs.

In an application of the MLP attack to the Splitting Attack on Interpose PUFs, we underlined the importance of adding MLP to the standard toolkit for PUF security evaluation and demonstrated that the low data complexity shown in this work has also applications in the security analysis of composite PUF designs.

We falsified claims of Mursi et al. [17] of extremely low data complexity of XOR Arbiter PUF modeling attacks and reevaluated their results, showing that their attack still has data complexity an order of magnitude lower than state-of-the-art attacks. We further falsified the claims of Nguyen et al. [19] that the Logistic Regression attack has the lowest data complexity among all modeling attacks on XOR Arbiter PUFs.

To facilitate future security analyses of PUFs and to avoid errors in such attacks, we publish all our implementations as a free open-source contribution to the pypuf framework [31], and encourage the PUF community to do similarly. The code of this work and the above detailed discussion of attack methodology, hyperparameter choices, and network design shed light on some of the inner workings of MLP-based PUF modeling attacks and may help to apply the MLP approach to other PUF designs.

Our results raise the question if reliability-based attacks like the recently presented one by Tobisch et al. [26] can benefit from the neural network approach in a similar way. If so, that may pave the way to generalize reliability attacks to PUF designs other than the XOR Arbiter PUF (and its variants).

Finally, with XOR Arbiter PUF, Interpose PUFs, and homogeneous XOR Feed-Forward Arbiter PUFs successfully attacked, it will be interesting to see which other PUF designs neural-network-based attacks will be able to model.

XI. Acknowledgements

The authors would like to thank Anita Aghaie, Ahmad O. Aseeri, Amir Moradi, Pranesh Santikellur, Johannes Tobisch, and the anonymous reviewers. We acknowledge the provided HPC computing time of Texas Tech University, Technische Universität Berlin, and Freie Universität Berlin.

The research was supported in part by the National Science Foundation under grant No. 2103563 and the German Ministry for Education and Research as BBDC 2 (ref. 01IS18025A).

References

[1] Anita Aghaie and Amir Moradi. Inconsistency of Simulation and Practice in Delay-based Strong PUFs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 520–551, July 2021.

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

[3] A. O. Aseeri, Y. Zhuang, and M. S. 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, July 2018.

[4] S. V. Sandeep Avvaru, Ziqing Zeng, and Keshab K. Parhi. Homogeneous and Heterogeneous Feed-Forward XOR Physical Unclonable Functions. *IEEE Transactions on Information Forensics and Security*, 15:2485–2498, 2020.

[5] Georg T. Becker. The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, Lecture Notes in Computer Science, pages 535–555. Springer Berlin Heidelberg, 2015.

[6] D. Chatterjee, D. Mukhopadhyay, and A. Hazra. PUF-G: A CAD Framework for Automated Assessment of Provable Learnability from Formal PUF Representations. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, November 2020.

[7] Durba Chatterjee, Urbi Chatterjee, Debdeep Mukhopadhyay, and Aritra Hazra. SACReD: An Attack Framework on SAC Resistant Delay-PUFs leveraging Bias and Reliability Factors. February 2021.

[8] J. Delvaux. Machine-Learning Attacks on PolyPUFs, OB-PUFs, RPUFs, LHS-PUFs, and PUF–FSMs. *IEEE Transactions on Information Forensics and Security*, 14(8):2043–2058, August 2019.

[9] Jeroen Delvaux and Ingrid Verbauwhede. Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium On*, pages 137–142. IEEE, 2013.

[10] Fatemeh Ganji, Shahin Tajik, Fabian Fäßler, and Jean-Pierre Seifert. Strong Machine Learning Attack Against PUFs with No Mathematical Model. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, Lecture Notes in Computer Science, pages 391–411. Springer Berlin Heidelberg, 2016.

[11] Blaise Gassend, Daihyun Lim, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Identification and authentication of integrated circuits. *Concurrency and Computation: Practice and Experience*, 16(11):1077–1098, September 2004.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017.

[13] Raghavan Kumar and Wayne Burleson. Side-Channel Assisted Modeling Attacks on Feed-Forward Arbiter PUFs Using Silicon Data. In Stefan Mangard and Patrick Schaumont, editors, *Radio Frequency Identification*, Lecture Notes in Computer Science, pages 53–67, Cham, 2015. Springer International Publishing.

[14] Yann Lecun, Leon Bottou, Genevieve B. Orr, and K. Muller. Efficient backdrop. In G. Orr and K. Muller, editors, *Neural Networks*. Springer, 1998.

[15] J. W. Lee, Daihyun Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*, pages 176–179, June 2004.

[16] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas. Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching. In *2012 IEEE Symposium on Security and Privacy Workshops*, pages 33–44, May 2012.

[17] Khalid T. Mursi, Bipana Thapaliya, Yu Zhuang, Ahmad O. Aseeri, and Mohammed Saeed Alkatheiri. A Fast Deep Learning Method for Security Vulnerability Study of XOR PUFs. *Electronics*, 9(10):1715, October 2020.

[18] Khalid T Mursi, Yu Zhuang, Mohammed Saeed Alkatheiri, and Ahmad O Aseeri. Extensive examination of xor arbiter pufs as security primitives for resource-constrained iot devices. In *2019 17th International Conference on Privacy, Security and Trust (PST)*, pages 1–9. IEEE, 2019.

[19] Phuong Ha Nguyen, Durga Prasad Sahoo, Chenglu Jin, Kaleel Mahmood, Ulrich Rührmair, and Marten van Dijk. The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 243–290, August 2019.

[20] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical One-Way Functions. *Science*, 297(5589):2026–2030, September 2002.

[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*, CCS '10, pages 237–249, New York, NY, USA, 2010. ACM.

[22] Pranesh Santikellur, Aritra Bhattacharyay, and Rajat Subhra Chakraborty. Deep Learning based Model Building Attacks on Arbiter PUF Compositions. page 10, 2019.

[23] Pranesh Santikellur, Lakshya, Shashi Ranjan Prakash, and Rajat Subhra Chakraborty. A Computationally Efficient Tensor Regression Network based Modeling Attack on XOR APUF. In *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6, December 2019.

[24] G. Edward Suh and Srinivas Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 9–14, New York, NY, USA, 2007. ACM.

[25] Shahin Tajik, Enrico Dietz, Sven Frohmann, Jean-Pierre Seifert, Dmitry Nedospasov, Clemens Helfmeier, Christian Boit, and Helmar Dittrich. Physical Characterization of Arbiter PUFs. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Camille Salinesi, Moira C. Norrie, and Óscar Pastor, editors, *Advanced Information Systems Engineering*, volume 7908, pages 493–509. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[26] Johannes Tobisch, Anita Aghaie, and Georg T. Becker. Combining Optimization Objectives: New Modeling Attacks on Strong PUFs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 357–389, February 2021.

[27] 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.

[28] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[29] Abilash Venkatesh, Aishwarya Bahudhanam Venkatasubramaniyan, Xiaodan Xi, and Arindam Sanyal. 0.3 pJ/Bit Machine Learning Resistant Strong PUF Using Subthreshold Voltage Divider Array. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(8):1394–1398, August 2020.

[30] Nils Wisiol, Georg T. Becker, Marian Margraf, Tudor A. A. Soroceanu, Johannes Tobisch, and Benjamin Zengin. Breaking the Lightweight Secure PUF: Understanding the Relation of Input Transformations and Machine Learning Resistance. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications*, Lecture Notes in Computer Science, pages 40–54, Cham, 2020. Springer International Publishing.

[31] Nils Wisiol, Christoph Gräbnitz, Christopher Mühl, Benjamin Zengin, Tudor Soroceanu, Niklas Pirnay, and Khalid T. Mursi. *pypuf: Cryptanalysis of Physically Unclonable Functions*, 2021.

[32] Nils Wisiol, Christopher Mühl, Niklas Pirnay, Phuong Ha Nguyen, Marian Margraf, Jean-Pierre Seifert, Marten van Dijk, and Ulrich Rührmair. Splitting the Interpose PUF: A Novel Modeling Attack Strategy. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 97–120, June 2020.

[33] Nils Wisiol, Khalid T. Mursi, Jean-Pierre Seifert, and Yu Zhuang. Neural-Network-Based Modeling Attacks on XOR Arbiter PUFs Revisited. Technical Report 555, 2021.

[34] Risa Yashiro, Takanori Machida, Mitsugu Iwamoto, and Kazuo Sakiyama. Deep-Learning-Based Security Evaluation on Authentication Systems Using Arbiter PUF and Its Variants. In Kazuto Ogawa and Katsunari Yoshioka, editors, *Advances in Information and Computer Security*, volume 9836, pages 267–285. Springer International Publishing, Cham, 2016.

[35] Meng-Day Yu, Matthias Hiller, Jeroen Delvaux, Richard Sowell, Srinivas Devadas, and Ingrid Verbauwhede. A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication. *IEEE Transactions on Multi-Scale Computing Systems*, 2(3):146–159, July 2016.

[36] 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.

[37] Haoyu Zhuang, Xiaodan Xi, Nan Sun, and Michael Orshansky. A Strong Subthreshold Current Array PUF Resilient to Machine Learning Attacks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(1):135–144, January 2020.