Neural-Network-Based Modeling Attacks on XOR Arbiter PUFs Revisited

Nils Wisiol[1], Khalid T. Mursi[2], Jean-Pierre Seifert[1] and Yu Zhuang[3]

[1] Technische Universität Berlin {nils.wisiol,jean-pierre.seifert}@tu-berlin.de
[2] College of Computer Science and Engineering, University of Jeddah, Jeddah 21959, Saudi Arabia; kmursi@uj.edu.sa
[3] Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA; yu.zhuang@ttu.edu

Abstract. By revisiting recent neural-network based modeling attacks on XOR Arbiter PUFs from the literature, we show that XOR Arbiter PUFs and Interpose PUFs can be attacked faster, up to larger security parameters, and with orders of magnitude fewer challenge-response pairs than previously known.
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.

Keywords: Physical Unclonable Function · Strong PUFs · Machine Learning · Modeling Attacks · XOR Arbiter PUF

1 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 [PRTG02], 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 a electrical signal or leaser beam. Further, it is assumed to be *physically unclonable*, meaning that it is inherently impossible to produce two identical tokens. The envisioned secret-free cryptography shall be based on this unique response behavior of each individual unclonable token.

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 any cryptographic application would collapse, just like in the case of a leaked secret key.

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 [GLC+04, RSS+10], 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, 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[1]. More recently, this approach was complemented by a modeling attack methodology based on general models not derived from physical insights [YMIS16, MZAA19, AZA18, MTZ+20, SLPC19, WMP+20] and models that are extensions of a physical model [SLPC19]. As such attacks naturally do not require exact modeling of the PUF under attack, they allow for rapid testing of PUF design ideas. Consequently, using general neural networks such as the multilayer perceptron as an analysis tool for strong PUF design has found some adoption.

For example, in a recent modeling attack on the Interpose PUF by Wisiol et al. [WMP+20], 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. [SBC19] 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 [ZXSO20] 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. [VVXS20] 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 XOR Arbiter PUF, an electrical PUF design 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 orders 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 falsify statements of Nguyen et al. [NSJ+19] made at CHES 2019.

2. We show that our results reduce the data complexity of the Splitting Attack on

[1] A notable exception is the modeling of Bistable Ring PUFs without any model assumptions by Ganji et al. [GTFS16].

the Interpose PUF, thus demonstrate 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 falsify a recently claimed very low data complexity of XOR Arbiter PUF modeling attacks [MTZ+20].

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 [WGM+21].

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. 4, we discuss and evaluate the Logistic Regression attack as a baseline for comparison of attacks in this work. In Sec. 5, 6, and 7, we replicate neural-network-based attacks from the literature and discuss design choices as well as attack performance, with a comparison of the attacks given in Sec. 8. We draw conclusions and discuss future work in Sec. 9.

2 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 [Bec15]. 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 [TAB21]. Chatterjee et al. [CCMH21] 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 the generic approaches used in this work, highly specialized attacks based on machine learning are also an essential part of the security analysis of PUFs, as demonstrated by Delvaux [Del19] in a collection of five attacks on PUFs with accompanying lightweight obfuscation logic.

Another branch of security analysis of PUFs uses provable methods in the PAC framework. In certain cases, this can be done without using a physically inspired model [GTFS16], which relates to the application of generic neural networks in this work. Recently, a semiautomated approach for analysis was proposed [CMH20].

Provable techniques have also been used by PUF designers to show a design to be secure, such as a proposal to disallow the attacker from querying a large number of challenges [YHD+16] or by basing the security of the PUF on a commonly used cryptographic hardness assumption [HRvD+17]. These approaches, however, require to enlarge the trusted computing base, which increases the attack surface of physical attacks.

Despite the ongoing setbacks in the design and implementation of secure PUFs, there is also some research on PUF implementations with respect to the security against side-channel information. Tajik et al. [TDF+14] attacked an Arbiter PUF implementation

using photonic emission analysis, confirming the accuracy of the delay-based Arbiter PUF model as a byproduct. More recently, Aghaie and Moradi [AM20] proposed a general technique to protect PUF implementations against attacks based on power side channel analysis.

3 Preliminaries

3.1 Additive Delay Model

n-bit k-XOR Arbiter PUFs [GLC+04, SD07] can be modeled using Boolean functions $f : \{-1, 1\}^n \to \{-1, 1\}$ parameterized by real values $W \in \mathbb{R}^{k \times n}$ and $b \in \mathbb{R}^k$, where 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 a Arbiter PUF circuit, can be found in Appendix A.

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 [RSS+10] (cf. Sec. 4).

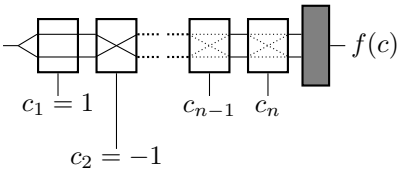
All modeling attacks in this work are based on simulated challenge-response data, instead of real-world data obtained from physical implementations of Arbiter PUFs. The reason for this is twofold. First, security analysis with a focus on the PUF design should be done on ideal data, i.e., without regard to the properties of a specific implementation. The 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 [DV13, TDF+14].

For the modeling of noise, we rely on the model by Delvaux et al. [DV13]. 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

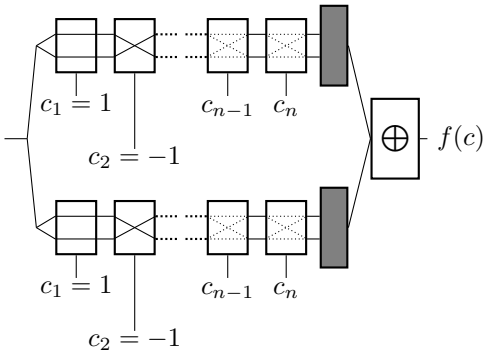
$$f(c) = \operatorname{sgn} \left(\langle W, x \rangle + b + N \right),$$

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 (see above and Appendix A). 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



(a) Schematic of an Arbiter PUF with n-bit challenge c (represented with bit values from $\{-1, 1\}$) and final response $f(c)$.



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

Figure 1: Schematics of Arbiter PUF and XOR Arbiter PUF.

same response when given the challenge twice in independent evaluations,

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

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.

3.2 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 [Bec15, TAB21].

In this work, however, we consider the weaker attacker model introduced 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, which is the case for many novel PUF designs. 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.

3.3 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 all attacks using the popular Keras framework and ran all experiments on the same CPU. 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 report 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 couple 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 [Del19]. Hence, instead of prediction accuracy, we report 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.

To make our work reproducible, we seed all involved random number generators with defined values.

4 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. [RSS+10] 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 [NSJ+19] and an attack on it [WMP+20]. As such, we use the results obtained with the LR attack as a baseline for comparison in this work.

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 Appendix A; 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 [TB15] 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 performance 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 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 faster convergence, 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 [KB17] instead of the originally used resilient backpropagation, as the later works poorly together with the use of mini batches.

Third, we apply the tanh activation function to each of the k delay values computed

Table 1: Empirical results on learning simulated XOR Arbiter PUFs obtained using our Keras-based implementation of the LR attack. Reference values of Tobisch and Becker [TB15] use up to 16 cores. (⋆ Result obtained using a different number of CRPs.)

n	k	CRPs	success rate	duration (max. threads)	memory	[TB15]
64	4	30k	10/10	<1 min @ 4		<1 min
64	5	260k	10/10	4 min @ 4		<1 min
64	6	2M	20/20	<1 min @ 4		1 min
64	7	20M	10/10	3 min @ 4		55 min
64	8	150M	10/10	28 min @ 4		391 min
64	9	500M	7/10	14 min @ 40	132 GiB	⋆2266 min
64	10	1B	6/10	41 min @ 40	197 GiB	-

by the respective arbiter chains, i.e. we change the model function from

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

This change was motiviated 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 analogon 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.

Even though our version of the LR attack technically does not fulfill the definition of logistic regression anymore, we will refer to this version as the *improved LR attack*. A sketch of the network structure used in the attack is displayed in Fig. 3d.

Using our Keras-based implementation together with these improvements, we could increase the performance of the LR algorithm (with respect to wall clock time), which is summarized in Tab. 1. 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. 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.

Our numbers confirm once more [RSS+10, WMP+20] 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}.

5 Tensor Regression Network Attack by Santikellur et al.

To reduce the computational effort for modeling attacks on XOR Arbiter PUFs, Santikellur et al. [SLPC19] 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 $b_{l,i} \in \mathbb{R}$. A drawing of the network structure is shown in Fig. 3c. 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 $w_{l,i}, b_{l,i}$, $l \in \{1, \ldots, k\}, i \in \{1, \ldots, R\}$. 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 weighed 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. 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. 2.

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. [SLPC19] 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. 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. We further discuss this in Sec. 8 and 9.

6 Multilayer Perceptron Attack by Aseeri et al.

After an attack on 3-XOR 64-bit Arbiter PUFs by Yashiro et al. [YMIS16] using a neural network with autoencoders and an attack by Alkatheriri et al. [AZ17] on Feed-Forward

Table 2: Comparing single-attempt attacks using rank R ECP-TRN versus R-attempt attacks using rank 1 ECP-TRN. 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. [SLPC19], in some cases we increased the number of CRPs to obtain any successful results.

n	k	[SLPC19] CRPs	our CRPs	R	attempts per run	total runs	run success rate
64	4	40k	40k	1	5	10	100%
64	4	40k	40k	5	1	10	100%
64	5	80k	320k	1	10	10	90%
64	5	80k	320k	10	1	10	90%
64	6	400k	800k	1	10	10	80%
64	6	400k	800k	10	1	10	80%
64	7	800k	800k	1	100	5	20%
64	7	800k	800k	1000	1	4	0%

Arbiter PUFs using a multilayer perceptron, Aseeri et al. [AZA18] were the first ones to attack XOR Arbiter PUFs of large size 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 [TB15].

Unfortunately, the source code of this attack was not immediately available after publication, and some attempts to replicate the work failed [SBC19]. This may be the reason this work so far found little attention within the PUF community and was not sufficiently considered in the security analysis of the Interpose PUF [NSJ+19].

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. 3, 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 7. A comparing overview can be found in Tab. 5; a drawing of the network can be found in 3a.

7 Multilayer Perceptron Attack by Mursi et al.

7.1 Neural Network used by Mursi et al.

In follow-up work to Aseeri et al. [AZA18] and Santikellur et al. [SBC19] (not to be confused with the ECP-TRN model), Mursi et al. [MTZ+20] presented an enhancement of

Table 3: Extended results on learning simulated XOR Arbiter PUFs obtained using our Keras-based implementation of the multilayer perceptron attack by Aseeri et al. [AZA18]. ⋆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.

			success			single core duration	
n	k	CRPs	rate	duration	memory	this work⋆	[AZA18]
64	4	400k	10/10	<1 min	<1 GiB	11 min	<1 min
64	5	400k	10/10	<1 min	<1 GiB	17 min	<1 min
64	6	2M	9/10	<1 min	1 GiB	8 min	7 min
64	7	5M	9/10	<1 min	2 GiB	20 min	12 min
64	8	30M	10/10	3 min	8 GiB	102 min	23 min
64	9	80M	9/10	86 min	29 GiB	3438 min	-
128	4	400k	10/10	<1 min	<1 GiB	17 min	1 min
128	5	3M	10/10	<1 min	2 GiB	33 min	5 min
128	6	20M	10/10	<1 min	10 GiB	28 min	19 min
128	7	40M	10/10	5 min	20 GiB	181 min	90 min
128	8	100M	1/10	45 min	50 GiB	1813 min	-

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 falsify 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 Fig. 3b. 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. We surmise that this benefits training of the network as it enables weight update for neurons that compute a negative value. While Santikellur et al. [SBC19] 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 [MTZ+20]. A detailed comparison of hyperparameters as used in the different attacks showed in this paper can be found in Tab. 5.

7.2 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, 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 [WGM+21] used throughout this work and in a recent LR-based attack on the Interpose PUF [WMP+20]. The CRP generation is, for performance reasons, based on the equivalent approach of using weights instead of delays. (For a formal proof and how to convert delays and weights, see Appendix A). 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. 4.

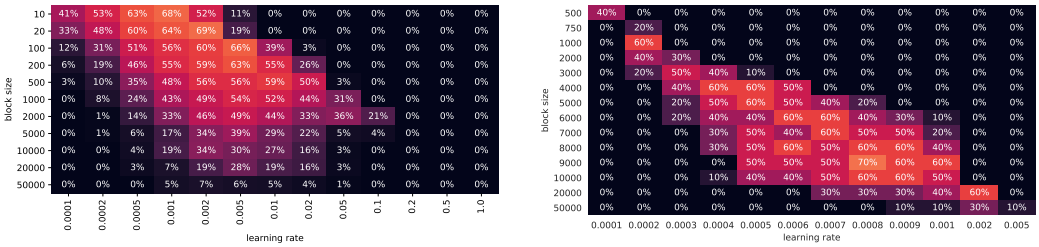
For challenge lengths 128 and 256, we found that the data complexity grows fast with the number of XORs. Nevertheless, for 128 bit challenges, it remains below the figures that Tobisch and Becker [TB15] reported for the LR attack; for 256 bit challenges we could not find numbers in the literature to compare to. However, the LR attack is known to have polynomially increasing data complexity in the challenge length [WMP+20]. 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 [NSJ+19].

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 [GLC+04, TDF+14, DV13].

7.3 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 4-XOR 64-bit and 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



(a) 4-XOR Arbiter PUF, 50k CRPs, 100+ runs each

(b) 8-XOR Arbiter PUF, 6M CRPs, 10 runs each

Figure 2: Success rate for training an 64-bit XOR Arbiter PUFs with the attack by Mursi et al. [MTZ+20], with varying learning rates and block sizes. For each combination, 10 learning attempts were run.

the learning process runs into the maximum number of epochs before a convergence could be achieved.

7.4 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. Note that the noise resilience is not a prerequisite for a successful attack for an active attacker, as they could also query a challenge multiple times and remove noise efficiently by majority voting the responses. We report detailed results on the noise resilience in Tab. 4.

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 [NSJ+19, WMP+20]. We found that, similar to the results we obtained on XOR Arbiter PUFs, the data complexity of the Splitting Attack can be significantly reduced. We were able to attack a 64-bit $(1, 7)$-Interpose PUF with 6 million CRPs in minutes, compared to 20 million CPRs and 20 hours required by the original implementation.

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 (Tab. 4) and conclude that the MLP attack is able to attack 64-bit (1,11) Interpose PUFs using 325M CRPs (Wisiol et al. [WMP+20] 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. [TAB21], the MLP attack provides better convergence rate and faster computation time.

These results demonstrate that the MLP-based attack is resilient against both label and 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.

Table 4: Empirical results on learning simulated XOR Arbiter PUFs obtained using our improved implementation of the neural network attack by Mursi et al. [MTZ+20]. 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 ("rel.").

rel.	n	k	CRPs	success rate	duration (max. threads)	memory	[MTZ+20] CRPs	[MTZ+20] duration
1.00	64	4	150k	10/10	<1 min @ 40	1 GiB	42k	<1 min
1.00	64	5	200k	10/10	<1 min @ 20	3 GiB	255k	2 min
1.00	64	6	2M	10/10	<1 min @ 40	2 GiB	680k	1 min
1.00	64	7	4M	10/10	<1 min @ 40	3 GiB	1.7M	5 min
1.00	64	8	6M	7/10	13 min @ 4		4.2M	9 min
1.00	64	9	45M	10/10	16 min @ 40	14 GiB		
1.00	64	10	119M	7/10	291 min @ 40	41 GiB		
1.00	64	11	325M	10/10	1898 min @ 40	104 GiB		
1.00	128	4	1M	9/9	<1 min @ 40	1 GiB		
1.00	128	5	1M	10/10	<1 min @ 40	2 GiB		
1.00	128	6	10M	9/10	<1 min @ 20	5 GiB		
1.00	128	7	30M	10/10	2 min @ 20	20 GiB		
1.00	256	4	6M	10/10	1 min @ 40	6 GiB		
1.00	256	5	10M	10/10	3 min @ 40	11 GiB		
1.00	256	6	30M	0/8	- @ 40	33 GiB		
1.00	256	7	100M	1/10	8 min @ 40	98 GiB		
0.85	64	4	180k	9/10	<1 min @ 4	0 GiB		
0.85	64	5	150k	10/10	<1 min @ 4	0 GiB		
0.85	64	6	2M	10/10	<1 min @ 4	1 GiB		
0.85	64	7	4M	9/9	3 min @ 4	2 GiB		

8 Comparison

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. [MTZ+20] could be traced back to an error in CRP generation, which leaves the work by Santikellur et al. [SLPC19] 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. [MTZ+20] 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. [AZA18] and Santikellur et al. [SBC19] (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.

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 medium size and large XOR Arbiter PUFs, but not for very small designs. Next to the above-mentioned steep increase of data complexity for larger challenge lengths of the MLP-attack (cf. Sec. 7), 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. As discussed above, we speculate that the ReLU activation function hampers weight updates during the backpropagation process and thus constitutes a disadvantage in the learning process.

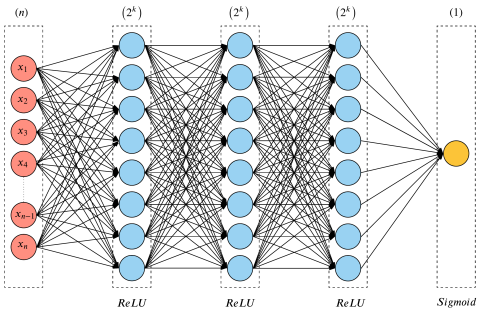
For a detailed comparison of the four modeling attacks, we provide graphs of the network structure in Fig. 3, display the chosen hyperparameters in Tab. 5, and provide an overview over the data complexities in Fig. 4.

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 multithreading, 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.

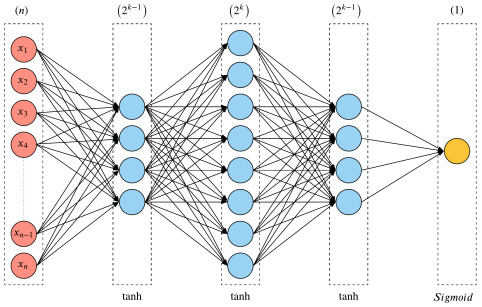
9 Conclusion

In this work, we compared three neural-network-based XOR Arbiter PUF modeling attacks [SLPC19, AZA18, MTZ+20] 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.

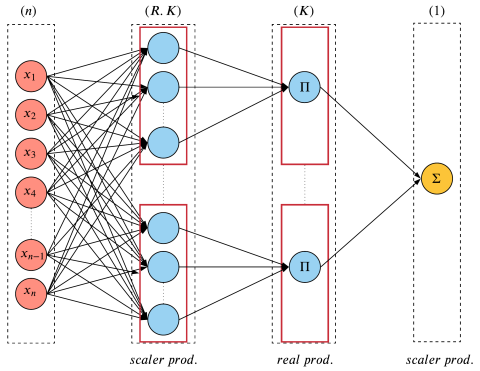
Providing the implementations of all four attacks studied in this work in a common framework and tested on a common platform, we are able to provide a fair comparison of their performances and confirm that previously reported better performances are not



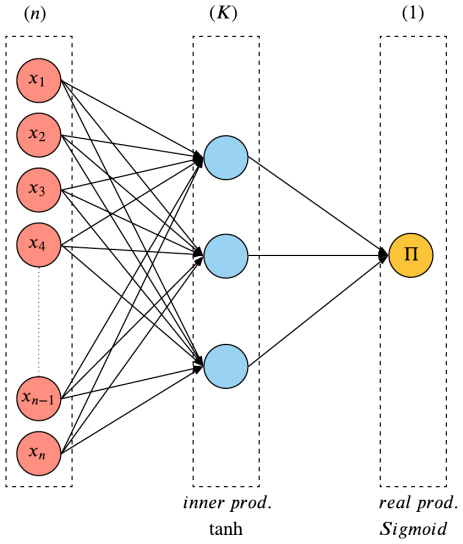
(a) The feed-forward neural network architecture for 3-XOR 64-bit using Aseeri's model [AZA18]. Since k is equal 3, the hidden layers consists of 8 neurons each. The architecture changes based on the number of streams in an k-XOR Arbiter PUF; the hidden layers use the ReLU activation function.



(b) The feed-forward neural network architecture for 3-XOR 64-bit using Mursi's model [MTZ+20]. Since k is equal 3, the first and third hidden layers consists of 4 neurons each, the second layer however possesses 8 neurons. The architecture changes based on the number of streams in an k-XOR Arbiter PUF; the hidden layers use the tanh activation function.



(c) The neural network architecture used by Santikellur et al. [SLPC19], which is a parallel structure containing R models as used in the LR attack (Fig. 3d) and computing the weighed sum of all outputs. Here shown for $R = 2$.



(d) The network architecture used by our improved version of the LR attack [RSS+10]. The network very closely follows inspirtation derived from a physical model of the XOR Arbiter PUF.

Figure 3: Neural network comparison for XOR Arbiter PUF modeling attacks.

Table 5: Original parameter comparison of machine Learning attack methods on XOR Arbiter PUFs. k refers to the number of Arbiter PUFs in the XOR Arbiter PUF under attack. For this work, we reimplemented all attacks to obtain a fair comparison. Note that our implementation of the LR attack [RSS+10] also includes changes that improve attack performance.

	Santikellur et al. **[SLPC19]**	Aseeri et al. **[AZA18]**	Mursi et al. **[MTZ+20]**	Rührmair et al. **[RSS+10]**
Library	tensorflow v1	sklearn	Keras	numpy
Method	TRN	MLP	MLP	LR
Architecture	many delay mod.	$(2^k, 2^k, 2^k)$	$(2^{k-1}, 2^k, 2^{k-1})$	delay model
Hid. lay. activ.	—	ReLU	tanh	—
Output activ.	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Optimizer	Adam	Adam	Adam	RProp
Loss function	BCE	BCE	BCE	BCE
Learning rate	(multiple)	10^{-3}	adaptive	RProp default
Initializer	Glorot Normal	Glorot Unif.	Gaussian	Gaussian

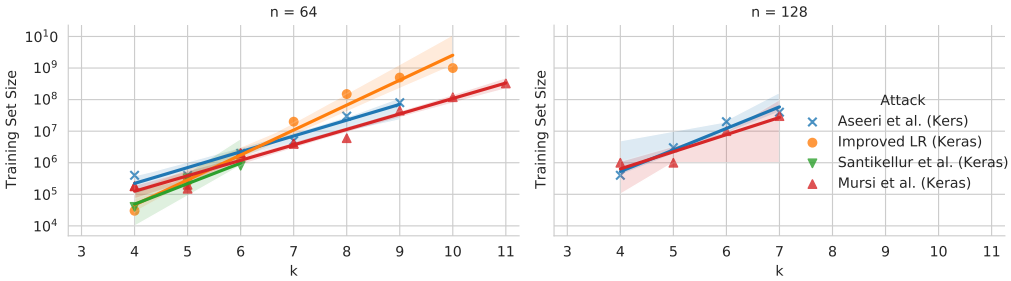


Figure 4: CRPs used by the attacks shown in this work for 64-bit Arbiter PUF with k individual arbiter chains. Our implementation of the attack by Mursi et al. outperforms the improved LR attack by an order magnitude with regard to data complexity. It improves on the attack by Aseeri et al. by cutting the number of required CRPs approximately in half.

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 [MTZ+20].

The MLP's advantage in data complexity 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 [WMP+20], 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 applications beyond the XOR Arbiter PUF.

We falsified claims of Mursi et al. [MTZ+20] 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. [NSJ+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 [WGM+21], 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. [TAB21] 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 Feed-Forward Arbiter PUFs [AZ17] successfully attacked, it will be interesting to see which other PUF designs neural-network-based attacks will be able to model.

Acknowledgements

The authors would like to thank Pranesh Santikellur for helpful comments on the TRN-based attack. We further thank Ahmad O. Aseeri for providing the source code of their attack and Johannes Tobisch for the helpful discussion of details on their LR implementation. Finally, we acknowledge the provided computing time of Technische Universität Berlin.

The research was supported in part by the National Science Foundation under Grant No. CNS-1526055.

References

[AM20] Anita Aghaie and Amir Moradi. TI-PUF: Toward Side-Channel Resistant Physical Unclonable Functions. *IEEE Transactions on Information Forensics and Security*, 15:3470–3481, 2020.

[AZ17] 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.

[AZA18] 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.

[Bec15] 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.

[CCMH21] 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.

[CMH20] 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.

[Del19] 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.

[DV13] 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.

[GLC+04] 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.

[GTFS16] 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.

[HRvD+17] C. Herder, L. Ren, M. van Dijk, M. Yu, and S. Devadas. Trapdoor computational fuzzy extractors and stateless cryptographically-secure physical unclonable functions. *IEEE Transactions on Dependable and Secure Computing*, 14(1):65–82, January 2017.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017.

[MTZ+20] 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.

[MZAA19] 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.

[NSJ+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.

[PRTG02] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical One-Way Functions. *Science*, 297(5589):2026–2030, September 2002.

[RSS+10] 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.

[SBC19] Pranesh Santikellur, Aritra Bhattacharyay, and Rajat Subhra Chakraborty. Deep Learning based Model Building Attacks on Arbiter PUF Compositions. page 10, 2019.

[SD07] 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.

[SLPC19] 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.

[TAB21] 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.

[TB15] 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.

[TDF+14] 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.

[VVXS20] 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.

[WGM+21] Nils Wisiol, Christoph Gräbnitz, Christopher Mühl, Benjamin Zengin, Tudor Soroceanu, and Niklas Pirnay. pypuf: Cryptanalysis of Physically Unclonable Functions, 2021.

[WMP+20] 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.

[YHD+16] 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.

[YMIS16] 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.

[ZXSO20] 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.

A Arbiter PUF Delays and Weights

An Arbiter PUF with n stages, once set up with a challenge $c \in \{-1, 1\}^n$, accumulates signal propagation delays for two electrical signals traveling though the stages of the Arbiter PUF. In stage i, the additional delays for the two signals are either d_i^{TT} and d_i^{BB} (if $c = -1$) or d_i^{TB} and d_i^{BT} (if $c = 1$). We define the total accumulated delay *after* stage i as d_i^{T} for the top output of that stage, and d_i^{B} for the bottom output of that stage. Additional delays are added at each stage, i.e.

$$d_i^{\mathrm{T}} = \begin{cases} d_{i-1}^{\mathrm{T}} + d_i^{\mathrm{TT}} & (c = -1)\,, \\ d_{i-1}^{\mathrm{B}} + d_i^{\mathrm{BT}} & (c = 1)\,, \end{cases}$$
$$d_i^{\mathrm{B}} = \begin{cases} d_{i-1}^{\mathrm{B}} + d_i^{\mathrm{BB}} & (c = -1)\,, \\ d_{i-1}^{\mathrm{T}} + d_i^{\mathrm{TB}} & (c = 1)\,. \end{cases}$$

The initial delays are zero, $d_0^{\mathrm{T}} = d_0^{\mathrm{B}} = 0$. The delay difference after the i-th stage is $\Delta D_i = d_i^{\mathrm{T}} - d_i^{\mathrm{B}}$; we abbreviate the delay difference after the n-th stage to $\Delta D_n = \Delta D$.

Theorem 1. *For $n \in \mathbb{N}$ and given delay values $d_i^{TT}, d_i^{TB}, d_i^{BT}, d_i^{BB} \in \mathbb{R}^+$ such that for all $c \in \{-1, 1\}$, for $1 \leq i \leq n$, there exists $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$*

$$\Delta D = \langle w, x \rangle + b,$$

where $x = (x_i)_i = \left(\prod_{j=i}^{n} c_j\right)_i$. *For even* n *we have*

$$w = \frac{1}{2}\begin{pmatrix} d_1^{TT} - d_1^{BB} - d_1^{BT} + d_1^{TB} \\ -d_2^{TT} + d_2^{BB} + d_2^{BT} - d_2^{TB} + d_1^{TT} - d_1^{BB} + d_1^{BT} - d_1^{TB} \\ d_3^{TT} - d_3^{BB} - d_3^{BT} + d_3^{TB} - d_2^{TT} + d_2^{BB} - d_2^{BT} + d_2^{TB} \\ \vdots \\ -d_{n-2}^{TT} + d_{n-2}^{BB} + d_{n-2}^{BT} - d_{n-2}^{TB} + d_{n-3}^{TT} - d_{n-3}^{BB} + d_{n-3}^{BT} - d_{n-3}^{TB} \\ d_{n-1}^{TT} - d_{n-1}^{BB} - d_{n-1}^{BT} + d_{n-1}^{TB} - d_{n-2}^{TT} + d_{n-2}^{BB} - d_{n-2}^{BT} + d_{n-2}^{TB} \\ -d_n^{TT} + d_n^{BB} + d_n^{BT} - d_n^{TB} + d_{n-1}^{TT} - d_{n-1}^{BB} + d_{n-1}^{BT} - d_{n-1}^{TB} \end{pmatrix},$$

$$b = 1/2\left(d_n^{TT} - d_n^{BB} + d_n^{BT} - d_n^{TB}\right);$$

Similar formulae exist for odd n.

Proof. The proof is by induction over i. For $i = 1$ we have

$$\Delta D_1 = d_1^{\mathrm{T}} - d_1^{\mathrm{B}} = \begin{cases} d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}} & (c_1 = -1 \iff 1/2 - 1/2 c_1 = 1), \\ d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}} & (c_1 = 1 \iff 1/2 - 1/2 c_1 = 0). \end{cases}$$

Using the fact that $1/2 - 1/2 c_1 \in \{0, 1\}$, we can write

$$\begin{aligned} \Delta D_1 &= (1/2 - 1/2 c_1)\left(d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}}\right) + (1 - (1/2 - 1/2 c_1))\left(d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}}\right) \\ &= (1/2 - 1/2 c_1)\left(d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}}\right) + (1/2 + 1/2 c_1)\left(d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}}\right) \\ &= c_1 \cdot \underbrace{1/2\left(d_1^{\mathrm{BB}} - d_1^{\mathrm{TT}} + d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}}\right)}_{w_1^{(1)}} + 1/2 \underbrace{\left(d_1^{\mathrm{TT}} - d_1^{\mathrm{BB}} + d_1^{\mathrm{BT}} - d_1^{\mathrm{TB}}\right)}_{b^{(1)}} \\ &= \left\langle x^{(1)}, w^{(1)} \right\rangle + b^{(1)} \end{aligned}$$

where $c^{(1)} = \left(c_1^{(1)}\right) \in \{-1, 1\}^1$, $x^{(1)} = \left(\prod_{j=i}^{1} c_i^{(1)}\right)_i = \left(x_1^{(1)}\right) = (c_1)$ and $w^{(1)} = \left(w_1^{(1)}\right) \in \mathbb{R}^1$.

Assuming $\Delta D_{i-1} = \left\langle x^{(i-1)}, w^{(i-1)} \right\rangle + b^{(i-1)}$, we have

$$\begin{aligned} \Delta D_i = d_i^{\mathrm{T}} - d_i^{\mathrm{B}} &= \begin{cases} d_{i-1}^{\mathrm{T}} + d_i^{\mathrm{TT}} - d_{i-1}^{\mathrm{B}} - d_i^{\mathrm{BB}} & (c_i = -1 \iff 1/2 - 1/2 c_i = 1), \\ d_{i-1}^{\mathrm{B}} + d_i^{\mathrm{BT}} - d_{i-1}^{\mathrm{T}} - d_i^{\mathrm{TB}} & (c_i = 1 \iff 1/2 - 1/2 c_i = 0), \end{cases} \\ &= \begin{cases} \Delta D_{i-1} + d_i^{\mathrm{TT}} - d_i^{\mathrm{BB}} & (c_i = -1 \iff 1/2 - 1/2 c_i = 1), \\ -\Delta D_{i-1} + d_i^{\mathrm{BT}} - d_i^{\mathrm{TB}} & (c_i = 1 \iff 1/2 - 1/2 c_i = 0), \end{cases} \end{aligned}$$

Using the fact that $1/2 - 1/2 c_i \in \{0, 1\}$, we can write

$$\begin{aligned} \Delta D_i &= (1/2 - 1/2 c_i)\left(\Delta D_{i-1} + d_i^{\mathrm{TT}} - d_i^{\mathrm{BB}}\right) + (1/2 + 1/2 c_i)\left(-\Delta D_{i-1} + d_i^{\mathrm{BT}} - d_i^{\mathrm{TB}}\right) \\ &= c_i \cdot 1/2 \left(\underbrace{-d_i^{\mathrm{TT}} + d_i^{\mathrm{BB}} + d_i^{\mathrm{BT}} - d_i^{\mathrm{TB}}}_{2\hat{w}^{(i)}} - 2\Delta D_{n-1}\right) - \underbrace{1/2\left(d_i^{\mathrm{TT}} - d_i^{\mathrm{BB}} + d_i^{\mathrm{BT}} - d_i^{\mathrm{TB}}\right)}_{\hat{b}^{(i)}} \\ &= c_i \cdot \left(\hat{w}^{(i)} - \Delta D_{i-1}\right) + \hat{b}^{(i)} \\ &= c_i \cdot \left(\hat{w}^{(i)} - \left\langle x^{(i-1)}, w^{(i-1)} \right\rangle - b^{(i-1)}\right) + \hat{b}^{(i)} \\ &= c_i \cdot \left(\hat{w}^{(i)} - \left(\sum_{l=1}^{i-1} w_l^{(i-1)} \prod_{j=l}^{i-1} c_j\right) - b^{(i-1)}\right) + \hat{b}^{(i)} \\ &= \left(\sum_{l=1}^{i-1} -w_l^{(i-1)} \prod_{j=l}^{i} c_j\right) + \left(\hat{w}^{(i)} - b^{(i-1)}\right) c_i + \hat{b}^{(i)}. \end{aligned}$$

Setting $\mathbb{R}^i \ni w^{(i)} = \left(-w_1^{(i-1)}, \ldots, -w_{i-1}^{(i-1)}, \hat{w}^{(i)} - b^{(i-1)}\right)$ and $b^{(i)} = \hat{b}^{(i)}$ we continue

$$\Delta D_i = \sum_{l=1}^{i} w_l^{(i)} \prod_{j=l}^{i} c_j + \hat{b}^{(i)} = \left\langle w^{(i)}, x^{(i)} \right\rangle + b^{(i)}.$$

Finally, for $i = n$, we set $w = w^{(n)}$,$x = x^{(n)}$, and $b = b^{(n)}$ and conclude that

$$\Delta D = \langle w, x \rangle + b.$$

Hence, an Arbiter PUF with positive delays $d_i^{\mathrm{TT}}, d_i^{\mathrm{TB}}, d_i^{\mathrm{BT}}, d_i^{\mathrm{BB}}$ for each stage i can be modeled as claimed. □

Corollary 1. *Assuming the delays $d_i^{TT}, d_i^{TB}, d_i^{BT}, d_i^{BB}$ for an Arbiter PUF are distributed according to a Gaussian with all the same mean and variance σ^2, the weights w and bias b of $\Delta D = \langle w, x \rangle + b$ are distributed according to*

$$w_0 \sim \mathcal{N}\left(0, \sigma^2\right), \quad w_i \sim \mathcal{N}\left(0, 2\sigma^2\right), \quad b \sim \mathcal{N}\left(0, \sigma^2\right).$$