

Modeling Attacks on State-of-the-art PUF Architectures

Zhuanhao Wu, Hiren Patel, Mahesh Tripunitara, Manoj Sachdev

This report provides details on the reproduction of previous machine learning attacks on delay-based PUFs.

1 Arbiter Based PUFs

The core of arbiter based PUFs is one or more chains of arbiters. In one of the chains, two electrical signals are fired simultaneously and delayed by the arbiters which are controlled by external binary signals. At the end of one chain, another arbiter, not controlled externally, determines which of the two signals arrives first. In arbiter based PUFs, a k -bit binary signal, or the challenge, acts as an input. After a transformation, possibly a unitary one, these signals control the arbiters. The outputs of each chain are then combined together to obtain the output of the PUFs.

1.1 Standard Arbiter PUFs

Standard arbiter PUFs are the simplest form of arbiter based PUFs. A standard arbiter PUF has only one chain of arbiters and the output of that chain is the output of the PUF. Figure 1 shows an N -bit standard arbiter PUF. $\delta_{0,i}$ represents the differences of delays between two output of the i -th arbiter, if the control signal $C[i] = 0$ and $\delta_{1,i}$ represents the case where the control signal $C[i] = 1$. If the D input to the final arbiter is faster than the clk signal, the output Q will be 1, otherwise, the output Q will be 0.

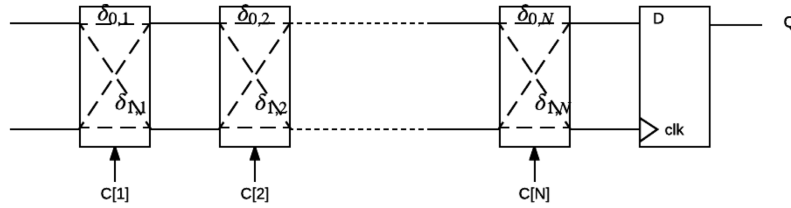


Figure 1: An N -bit standard arbiter PUF.

1.2 XOR Arbiter PUFs

An XOR arbiter PUF consist of l chains of standard arbiter PUFs, each has k bits. The challenge bits are the same for all of the standard arbiter PUFs and the output bits of each chain are XORED to together to form the single output of the PUF. Figure 2 shows the architecture of XOR arbiter PUFs conceptually. Each *Arb-PUF* in Figure 2 represents one chain of standard arbiter PUF.

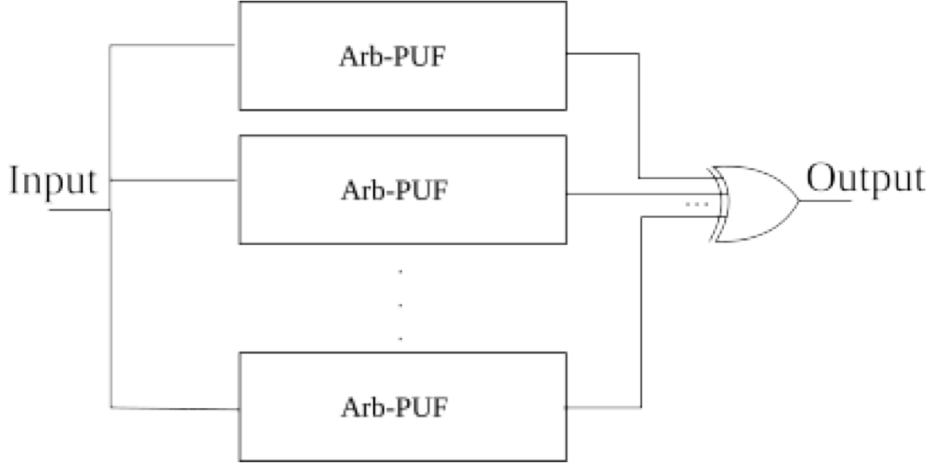


Figure 2: An illustration of XOR Arbiter PUF.

1.3 Lightweight Secure PUFs

A lightweight secure PUF, like an XOR arbiter PUF, consists of l chains of standard arbiter PUFs, each has k bits. However, the challenge bits go through different transformations for each chain. Also, the output bits of each chain are XORED to obtain multiple output bits. In this technical report, we only consider one output bit. To be specific, if we denote the control signal of the i -th arbiter in a standard arbiter PUF as c_i . Then it is defined in Equation 1 as:

$$\begin{aligned} c_{\frac{k+i+1}{2}} &= d_i, \text{ for } i = 1 \\ c_{\frac{i+1}{2}} &= d_i \oplus d_{i+1}, \text{ for } i = 1, 3, 5, \dots, k-1 \\ c_{\frac{k+i+2}{2}} &= d_i \oplus d_{i+1}, \text{ for } i = 2, 4, 6, \dots, k-2. \end{aligned} \quad (1)$$

The $\oplus$ represents binary XOR. The input challenge bits to the lightweight secure PUF are shifted for each of the l chains of standard arbiter PUFs to obtain d_i .

However, the attacking technique for dealing with multiple output bits is discussed in [1], which assumes that the single output bit attack is possible.

2 DCDE Based PUFs

In arbiter based PUFs, we use arbiters as the delay elements and exploit the manufacturer variations to obtain randomness. In a DCDE based PUF, we add Digitally Controlled Delay Elements [2], or DCDEs, before each of the arbiters. The control signals of the DCDEs are transformed from the input challenge bits.

3 Machine Learning Modeling

Throughout this technical report, we consistently use k as the number of challenge bits and l as the number of chains if there are multiple chains inside a PUF architecture.

We use Logistic Regression (LR) in our experiments to adjust the parameters in the model so that the model outputs the training set with maximal likelihood, as is done in [3]. Given a k -bit challenge $C = c_1c_2\dots c_k$, we apply some model parameterized by w to the challenge bits and get an output $f_w(C)$. We then assign a probability distribution to the output, $p(C, t|w) = \sigma(tf_w(C))$, representing the probability of our model outputting t given challenge bits C . The probability of model f_w outputting the training set M of CRPs is given in Equation 2.

$$P(M|w) = \prod_{(C,t) \in M} p(C, t|w) = \prod_{(C,t) \in M} \sigma(tf_w(C)). \quad (2)$$

Equation 3 gives the maximal likelihood estimates (MLE) of the parameter w .

$$w^* = \underset{w}{\operatorname{argmin}} (-\ln P(M|w)) = \underset{w}{\operatorname{argmin}} \sum_{(C,t) \in M} -\ln(\sigma(tf_w(C))) \quad (3)$$

For different PUFs, we provide different f_w as models. Equation 4 presents the model for standard arbiter PUFs.

$$f_w(C) = \operatorname{sgn}(\Delta) = \operatorname{sgn}(w^T \Phi) \quad (4)$$

In Equation 4, $w \in \mathbb{R}^{(k+1) \times 1}$ captures the characteristics of the PUF. $\Phi \in \mathbb{R}^{(k+1) \times 1}$ is the vector of pre-processed challenge bits, defined in Equation 5. $\Delta = w^T \Phi$ is the difference of delays between two paths of the arbiter PUF.

$$\Phi(j) = \begin{cases} (-1)^{c_j} \Phi(j+1) & j \leq k, \\ 1 & j = k+1 \end{cases} \quad (5)$$

Equation 6 gives the model for XOR arbiter PUFs:

$$f_w(C) = \prod_{i=1}^l \operatorname{sgn}(\Delta_i) = \prod_{i=1}^l \operatorname{sgn}(w_i^T \Phi_i) = \operatorname{sgn}(\prod_{i=1}^l w_i^T \Phi_i) \quad (6)$$

In Equation 6, Φ_i is the pre-processed challenge bits of the i -th arbiter PUF chain and w_i is the parameter vector of the i -th arbiter PUF chain.

The model for Lightweight Secure PUFs are the same for XOR arbiter PUFs, but the preprocessing of challenge bits is different. We supplied different challenge bits

to different chains of the model, according to the input network of Lightweight Secure PUFs [4].

We use the same model as Lightweight Secure PUFs for attacking DCDE PUFs.

3.1 Model Training

We applied RProp algorithm[5], which is a variant of gradient descent algorithm, to optimize the model. We referred to the implementation at <http://www.pcp.in.tum.de/code/lr.zip> and implemented this algorithm in **tensorflow**.

4 Experiment Setup

The experiments with modeling can be divided into three phases. In the first phase or the simulating phase, the environment initializes different PUFs and collect challenge-response pairs. In the second phase or the training phase, the environment feeds the challenge-response pairs to the machine learning models to learn the behavior of the PUFs. In the third phase or the test phase, the environment feeds a test set of challenge-response pairs to the trained model and collect various performance metrics of models.

4.1 Software Framework

We use **numpy**¹ to simulate the delays inside the PUFs. The delays of the PUFs are simple to calculate, and thus **numpy** provides us with the flexibility of trying different architectures of the PUFs and enough efficiency. For training and test phases, we use **tensorflow**², a general machine learning framework. The training phase requires multiple scans of the CRP set and optimization of the model, thus it requires a large number of computations. With **tensorflow**, we can easily accelerate the computations on different backends such as GPUs and CPU clusters.

4.2 Methodology

We first reproduced of previous work of modeling attacks and then we conducted similar modeling attacks on new PUF architectures.

In order to reproduce previous results on modeling arbiter-based PUFs, we followed [3]. According to [3], we trained the parameters in LR using RProp algorithm. However, we were not able to obtain the same misclassification rate graphs when we applied the same stop condition of training, which states that the training process should stop after the accuracies of 2 consecutive blocks of 5 iterations do not increase. Instead, we are able to obtain the same result if we set the prediction rate on the training set to be 99.9% for standard arbiter PUFs and 99% for XOR arbiter PUFs and Lightweight Secure PUFs. When training the models of XOR arbiter PUFs and Lightweight Secure PUFs, we also restart the training after the sum of the gradients is less than or equal to 0.0001.

¹<http://www.numpy.org/>

²<https://www.tensorflow.org/>

To evaluate the new DCDE based PUF architecture, we employ similar techniques used in arbiter based PUFs: the CRPs collected from the DCDE based PUFs are directly fed into the same model as is used in XOR arbiter PUFs and Lightweight Secure PUFs.

All figures are averaged over 5 experiments.

5 Results

5.1 Reproduction of Previous Work

5.1.1 Standard Arbiter PUFs

Figure 3 shows our evaluation of standard arbiter PUFs against LR attack. The blue line given by $y = 0.5137x^{-1.0}$ in Figure 3 is from in [3].

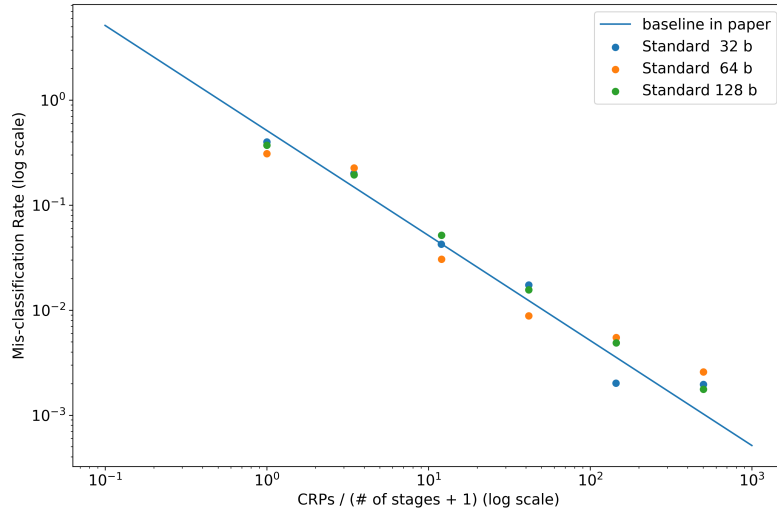


Figure 3: Mis-classification rate on standard arbiter PUFs.

5.1.2 XOR Arbiter PUFs

Figure 4 shows our evaluation of XOR arbiter PUFs against multiplication model. The blue line given by $y = 0.577x^{-1.007}$ in Figure 4 is from [3].

We also conducted experiments on the training time needed for modeling XOR arbiter PUFs. The results are shown in Table 1.

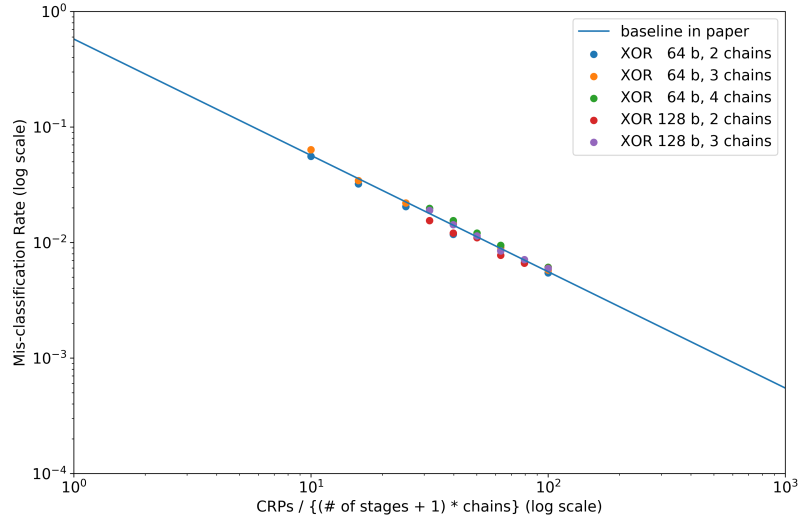


Figure 4: Mis-classification XOR arbiter PUFs.

Table 1: Training time and number of trials of XOR arbiter PUFs.
Training times and number of trials are averaged over different PUF instances.

# of stages	# of chains	# of CRPs	Training Time	# of Trials
64	4	12000	3:39 min	5.2
64	5	80000	1:10 hrs	5.4

5.1.3 Lightweight Secure PUFs

We conduct the experiment on the training time for Lightweight Secure PUFs, the results are shown in Table 2.

Table 2: Training Time and Number of Trials of Lightweight Secure PUFs.
Training times and number of trials are averaged over different PUF instances.

# of stages	# of chains	# of CRPs	Training Time	# of Trials
64	3	6000	2s	1
64	4	12000	1:30 hrs	106
128	3	15000	52s	1.4
128	4	500000	1:29 hrs	1.4

5.1.4 Evaluation of XOR Arbiter PUFs with Signal Swapping

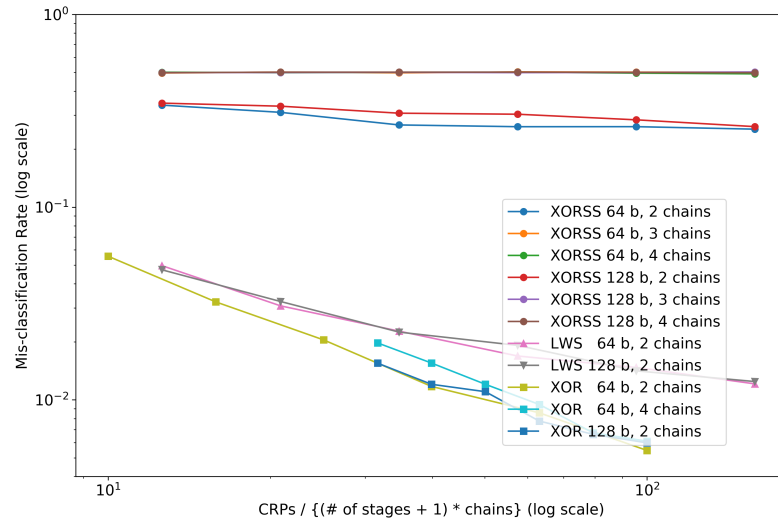


Figure 5: Mis-classification rate on XOR arbiter PUFs with signal swapping.

5.2 Evaluation of DCDE Based PUFs

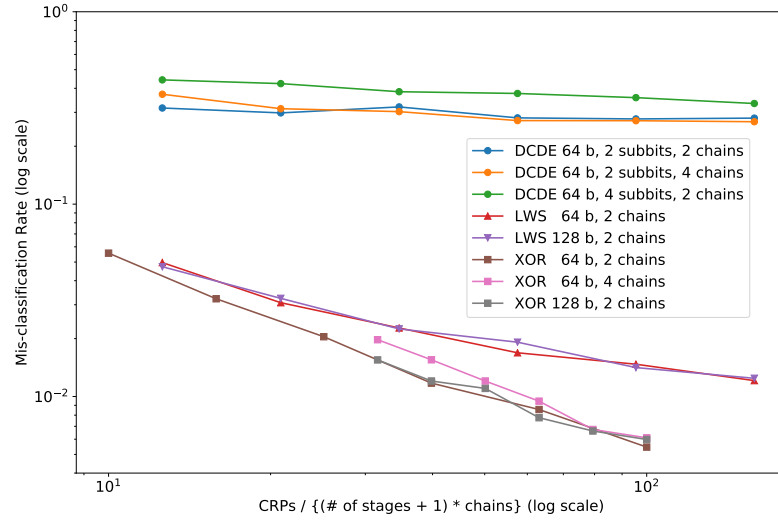


Figure 6: Mis-classification rate on DCDE based PUFs.

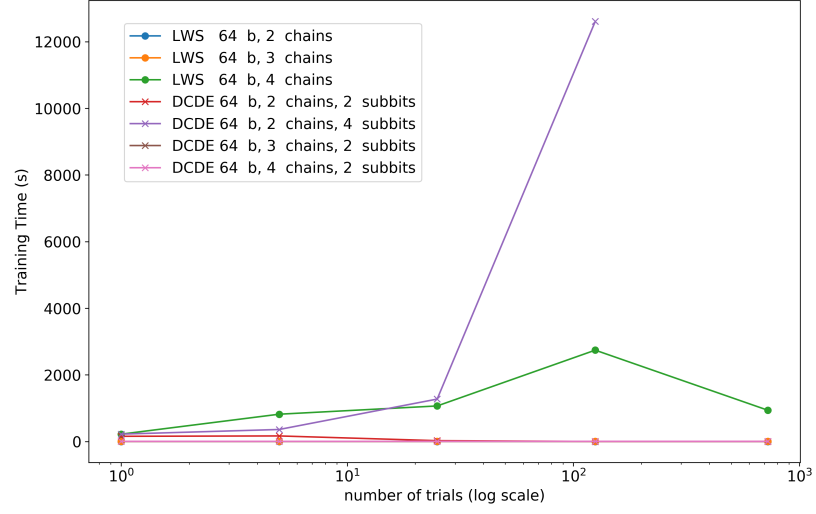


Figure 7: Training time of DCDE based PUFs, accuracy at 70%.

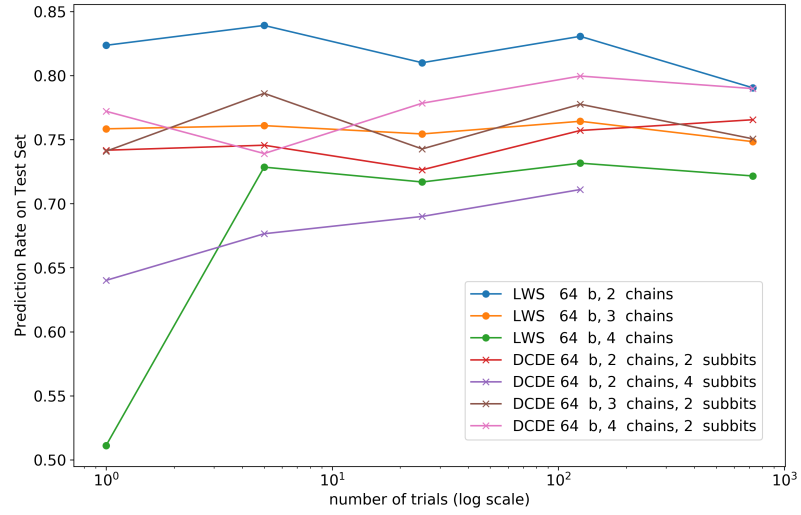


Figure 8: Mis-classification rate on test set of DCDE based PUFs, accuracy at 70%.

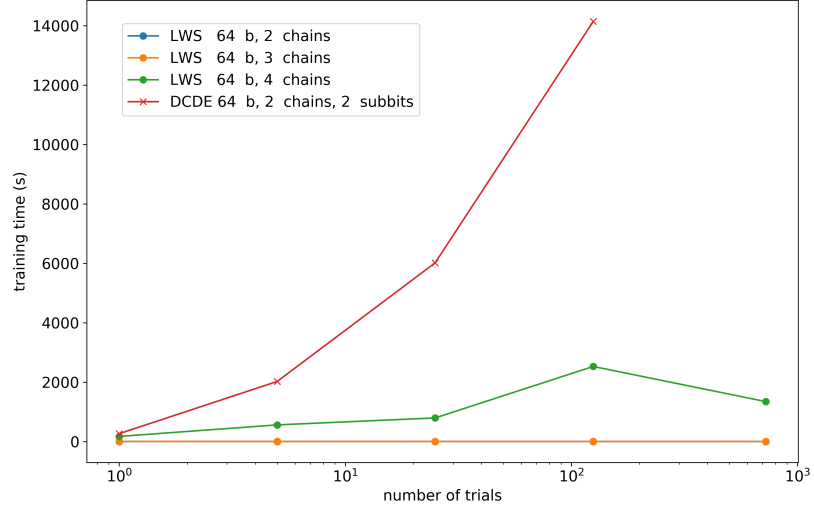


Figure 9: Training time of DCDE based PUFs, accuracy at 99%.

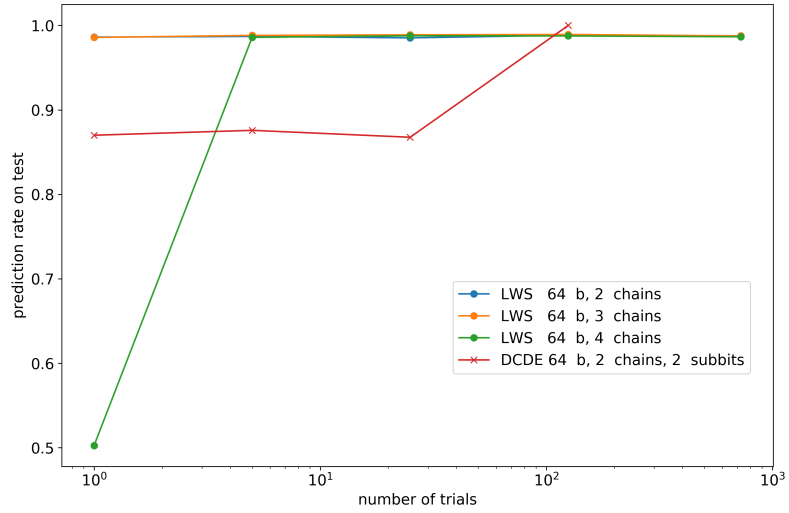


Figure 10: Mis-classification rate on test set of DCDE based PUFs, accuracy at 99%.

References

- [1] D. P. Sahoo, P. H. Nguyen, D. Mukhopadhyay, and R. S. Chakraborty, “A case of lightweight puf constructions: Cryptanalysis and machine learning attacks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, pp. 1334–1343, Aug 2015.
- [2] M. Maymandi-Nejad and M. Sachdev, “A digitally programmable delay element: design and analysis,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, pp. 871–878, oct 2003.
- [3] U. Ruhrmair, J. Solter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, “PUF modeling attacks on simulated and silicon data,” *IEEE Transactions on Information Forensics and Security*, vol. 8, pp. 1876–1891, nov 2013.
- [4] M. Majzoobi, F. Koushanfar, and M. Potkonjak, “Lightweight secure pufs,” in *2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 670–673, Nov 2008.
- [5] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: the RPROP algorithm,” in *IEEE International Conference on Neural Networks*, IEEE.