Modeling Arbiter PUFs: Development Project

Jakub Vogel
Dept. of Electrical and Computer Engineering
Rutgers University-New Brunswick
New Brunswick, USA
jakub.vogel@rutgers.edu

***Abstract*—Physical unclonable functions (PUFs) are hardware devices that produce unique responses to challenges to authenticate users. However, it has been demonstrated that these devices are vulnerable to machine learning modeling attacks in which correct responses to challenges can be predicted with high accuracy. This development project recreates previously successful modeling techniques and assesses their capability on up to 5-XORed arbiter PUFs by measuring each model's accuracy and training time. It demonstrates that deep learning models are the most effective method and can learn each APUF composition whereas simpler models like logistic regression and support vector machines are successfully defended by XORing APUFs.**

Keywords*—*physical unclonable functions, machine learning, deep learning, modeling attacks

I. Introduction

Physical unclonable functions (PUFs) are physical devices that rely on process variability to create functions that are unique, reliable, and not reproducible. PUFs accept a number of bits as a challenge and respond with a unique output. This process can be used to authenticate a user, as it is presumed that only the owner of the PUF would be able to answer the challenge correctly. Thus, it is essential that a PUF's response cannot be predicted accurately so that malicious users cannot be authenticated. However, machine learning (ML) based modeling attacks have proven successful in accurately predicting correct responses when given challenges [1]. As a result, a common countermeasure to modeling attacks is XORing. XORed PUFs XOR the responses of multiple PUFs to increase the complexity of the PUF and decrease susceptibility to modeling attacks.

This development project focuses on arbiter PUFs (APUFs)—a specific type of PUF that relies on chained multiplexers to output unique responses—and XORed APUFs. This project will test four different models: basic logistic regression (LR), logistic regression with RProp gradient descent, support vector machines (SVMs), and deep learning to examine the susceptibility of these PUFs to modeling-based attacks. Each model will be tested on a set of 64-bit challenge-response pairs (CRPs) for an APUF, 2-XOR APUF, 3-XOR APUF, 4-XOR APUF, and a 5-XOR APUF, measuring the prediction accuracy and training time for each combination.

II. Technical Background

[1] is a 2010 seminal work that analyzed the success of modeling attacks on PUFs. They utilized different ML strategies with logistic regression as their primary technique and successfully attacked 64-bit APUFs with up to six XORs. They note that their logistic regression model using RProp gradient descent was their most successful. This serves as the motivation for testing basic logistic regression, RProp logistic regression, and SVM models to try and achieve similar accuracy on XORed APUFs. More recently, in 2019, [2] analyzed newer deep learning-based modeling attacks on APUFs. These attacks were able to successfully model 64-bit APUFs with up to 6 XORs as well, but at a fraction of the time of [1] (20 minutes versus 31 hours for a 6-XOR APUF). This project will attempt to recreate the modeling techniques found in [1] and [2] and assess them on consumer hardware.

III. System Setup

All experiments are conducted on the same consumer laptop containing an Intel i5-9300H processor and 16GB of RAM. The models are built and assessed on Python 3.11.2 with the required packages listed in [3].

IV. Implementation

The basic logistic regression model is built on scikit-learn's [4] logistic regression, which solves using the Newton-Cholesky method. It terminates learning when it reaches the stopping criteria of 0.0001.

The RProp logistic regression model borrows from [5], which performs logistic regression from scratch but is modified to use RProp gradient descent [6] loosely based on the RProp pseudocode found in PyTorch [7]. It terminates learning early when either the loss increases or when the change in the loss is less than the tolerance of 0.0001. If it does not terminate early, it will run for 1000 epochs. The RProp-specific parameters (e.g., the maximum and minimum step sizes) are the same as recommended in [6].

The SVM model is built on scikit-learn's [4] implementation of SVM called support vector classification (SVC). However, due to classical SVC's struggle to scale as the number of data points grows, the model will switch to scikit-learn's linear SVC algorithm when the number of training points exceeds 50,000. The classical SVC is set to have a cache size of 2GB to improve its performance further and will train until convergence, whereas the linear SVC will perform a maximum of 2,000 iterations.

The deep learning model utilizes the Keras [8] framework in Tensorflow [9] to build a basic feedforward neural network. It consists of an input layer, a 12-node hidden layer using the

ReLU activation function, an 8-node hidden layer using the ReLU activation function, and an output layer using the sigmoid function. It uses the Adam optimizer and assesses its loss using binary cross-entropy. This model will terminate its training early if the loss fails to change by more than the 0.0001 tolerance and otherwise will perform 300 epochs.

The CRP dataset used to evaluate each model is sourced from [2]. It consists of 500,000 64-bit challenges with single-bit responses. For each different PUF (i.e., the APUF, 2-XOR APUF, 3-XOR APUF, 4-XOR APUF, and 5-XOR APUF), each model is evaluated using 5,000, 10,000, 20,000, 50,000, 100,000, and 500,000 CRPs with 80% of the CRPs used for training and the rest for testing. The resulting accuracy and training time are recorded for each combination. Accuracy is defined as the percentage of responses in the test set correctly predicted, and training time is defined as the wall clock time of training.

V. Evaluation

Fig. 1 and Fig. 2 show the accuracy and training time of the LR model versus the number of CRPs used. The LR model performs incredibly well on the APUF, achieving above 98% accuracy in less than 5 seconds of training time. However, it fails to successfully learn any of the XORed PUFs, struggling to achieve an accuracy greater than 65%.

The RProp LR model in Fig. 3 and Fig. 4 shows similar results to the basic LR model, with good results on the APUF but poor results on the XORed APUFs.

Fig. 5 and Fig. 6 show the SVM model, which performed better than either LR model achieving higher accuracies on both the APUF and 2-XOR APUF. It did so in considerably more time, taking around 200 seconds for 40k CRPs, whereas LR did so in under a second. Interesting to note here is the reduction in accuracy that occurs when going from 40k to 80k CRPs. This is likely attributed to the SVM model switching to a linear classifier when the number of points increases past 50k. It can be reasonably extrapolated that for the 2-XOR APUF, we would expect the accuracy of the classical SVM to keep improving, but the training time would take considerably longer as well (during testing, the model ran for 30 minutes without convergence before the linear SVM was used).

Fig. 7 and Fig. 8 show the deep learning model's performance. Perhaps unsurprisingly, it was the strongest model out of any tested and could successfully learn (with greater than 90% accuracy) any of the PUFs with a sufficient number of CRPs. It also did so in less than 3 minutes for all cases.

This work's results contrast with the results found in [1]. The authors there were able to successfully learn up to 5-XOR APUFs using LR, whereas the LR models used here were not. This is likely attributed to the different datasets used. When I ran the author's models on their own dataset, I was able to reproduce their high-accuracy results. However, on the new data, their methodology was also unable to learn this set of XORed APUFs. The more modern dataset from [2], which gathers CRPs from simulation models, might be more complex and better represent true PUFs than the simulation methods used in [1].

On the other hand, this work supports the findings of [2]. They, too, were able to train deep learning models on up to 5-XORed APUFs with high accuracy in relatively short amounts of time.

VI. Reproducibility

The source code can be found at [3]. A Python 3.11 installation will be needed along with the required packages to reproduce these findings. Executing the main script will run the same evaluations performed in this project and write the results to a CSV file. These results can be displayed in graphical format using the included results notebook.

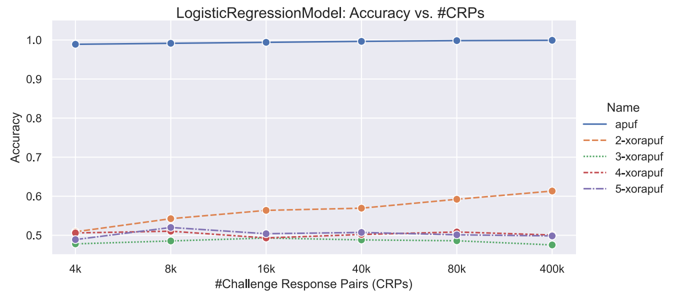

Fig. 1. Basic logistic regression model accuracy versus CRPs.

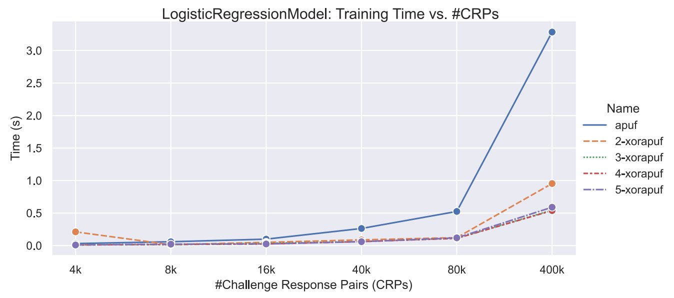

Fig. 2. Basic logistic regression model training time versus CRPs.

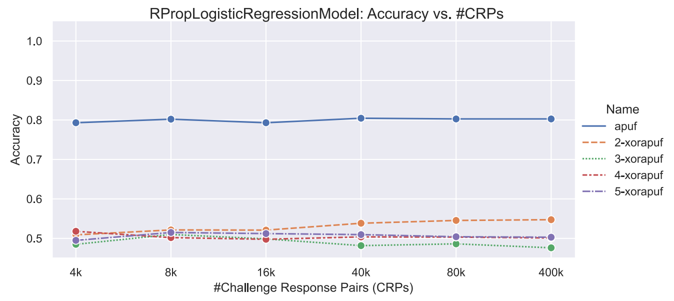

Fig. 3. RProp logistic regression model accuracy versus CRPs.

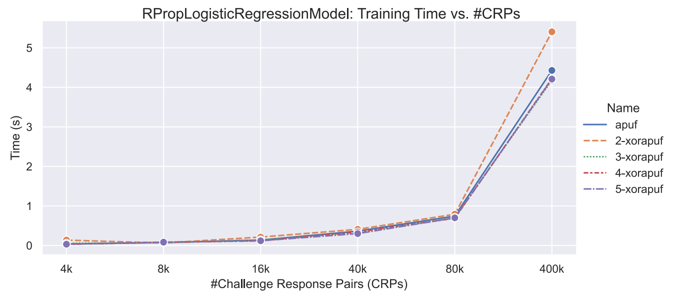

Fig. 4. RProp logistic regression model training time versus CRPs.

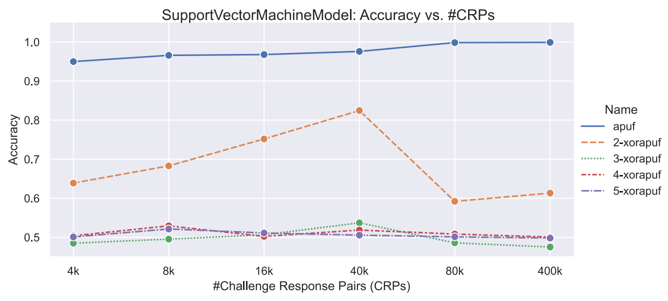

Fig. 5. SVM model accuracy versus CRPs.

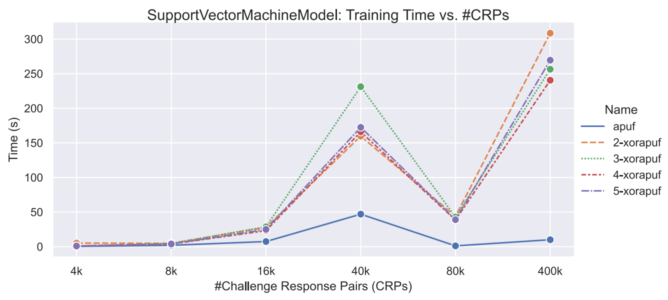

Fig. 6. SVM model training time versus CRPs.

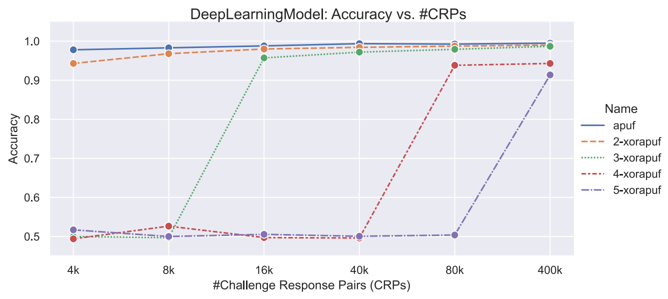

Fig. 7. Deep learning model accuracy versus CRPs.

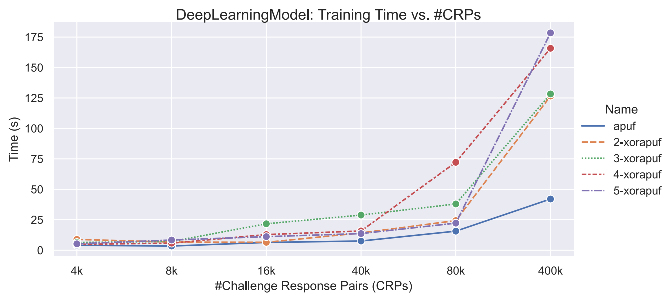

Fig. 8. Deep learning model training time versus CRPs.

ACKNOWLEDGMENT

I thank Dr. Sheng Wei for guidance.

REFERENCES

[1] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, “Modeling attacks on physical unclonable functions,” *Proceedings of the 17th ACM conference on Computer and communications security*, 2010.

[2] P. Santikellur, A. Bhattacharyay, R. S. Chakraborty, “Deep learning based model building attacks on arbiter PUF compositions,” *Cryptology ePrint Archive*, 2019.

[3] J. Vogel, “modeling-PUF,” https://github.com/JakubGV/modeling-PUFs.

[4] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] C. Hansen, “Logistic-Regression-From-Scratch,” https://github.com/casperbh96/Logistic-Regression-From-Scratch.

[6] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: The RPROP algorithm,” *IEEE International Conference on Neural Networks*, 1993.

[7] A. Paszke, *et al*., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates, Inc., 2019.

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

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