PUF Transformer: A transformer based modeling attack on Physical Unclonable Functions (PUF)[*]

Rhodel Quizon[†]
University of the Philippines Diliman
Diliman, Quezon City
Metro Manila, Philippines
rrquizon1@up.edu.ph

ABSTRACT

Physical Unclonable functions (PUFs) are one of the proposed hardware security devices for cryptographic applications. It utilized generating unique responses across devices from different N bit challenges. PUFs utilize the variation in manufacturing to show achieve unique responses.

Few studies have explored using different deep learning techniques to attack PUFs. In this paper we propose a shallow transformer based neural network to model PUFs. By using positional encoding, encoder layer composed of multihead self-attention and a linear layer, and 4 layer MLP head, 2-XOR Arbiter PUF was attacked. Using our transformer based model we are able to achieve 97.4%, 93.69% and 79.57% accuracy with 2-XOR, 3-XOR, 4-XOR 32 bit PUF tested over large number of datasets respectively. The network was also tested on 2-XOR 64-bit configuration tested to have an accuracy of 92.66%.

Categories and Subject Descriptors

Hardware Security [**Physical Unclonable Functions**]: XOR-Arbiter PUFs; Deep Learning [**Transformer Networks**]: Modeling Attack—*Hardware Hacking*

General Terms

Deep Learning, PUFs, Hardware Hacking

Keywords

Deep Learning, PUFs, Hardware Hacking

1. INTRODUCTION

Physical Unclonable Functions

Physical Unclonable Functions or PUFs are one of the recent emerging protocols in cryptographic protocols for IoT authentication. It utilizes the manufacturing variations to receive different responses from devices of same designs. It mostly uses variability in production such as delays to receive different responses. These means that PUFs are designed so that two devices with similar designs will have different group of challenge response pairs.

One of these PUFs are the arbiter PUFs which are composed of cascaded multiplexers with a D flip-flop at the end. This kind of PUF utilizes the delays propagated by different property of silicon to induce different response given similar design and similar inputs.Figure 1 shows the sample set-up of arbiter PUFs taken from [1] with 64 bit input challenge with one bit response.

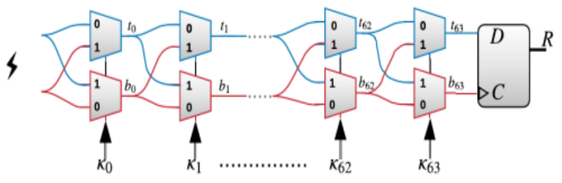

Figure 1: Sample Arbiter PUF circuit

The difference in manufacturing variations and complexity of the design can factor in different responses output at the end. To introduce more complexity and secure architecture, a new type of PUFs are designed in which multiple arbiter PUFs with outputs are connected to an XOR gate. The output of the XOR gate will then give the response output. Figure 2 shows a 3-XOR PUF. It has 3 arbiter PUFS connected to the XOR gate which gives the output response. Adding more arbiter PUFs to the XOR Arbiter PUF and adding more bits for the input will give more complexity to the design thus harder for a model to generalize.

Deep Leaning

Artificial intelligence has been having significant progress in the recent years. Deep learning has been used in solving different problems such as object detection, natural language processing, or image generation. One of the most recent state of the art architecture is the Transformer[2].

*

†

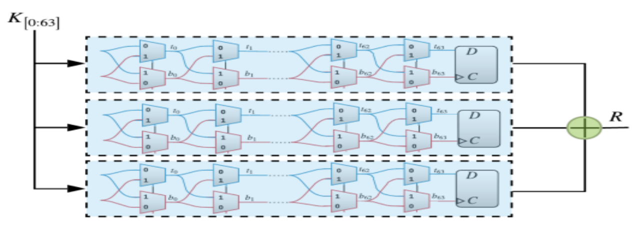

Figure 2: Sample 3-XOR 64 bit input Arbiter PUF

Transformer is one of the recent innovations in machine learning, originally used in natural language processing. It uses the concept of multihead attention to understand relationship between sequences. Figure 3 shows the full architecture of the original transformer. Additionally, positional encoding and linear layers are also used as part of the original architecture.

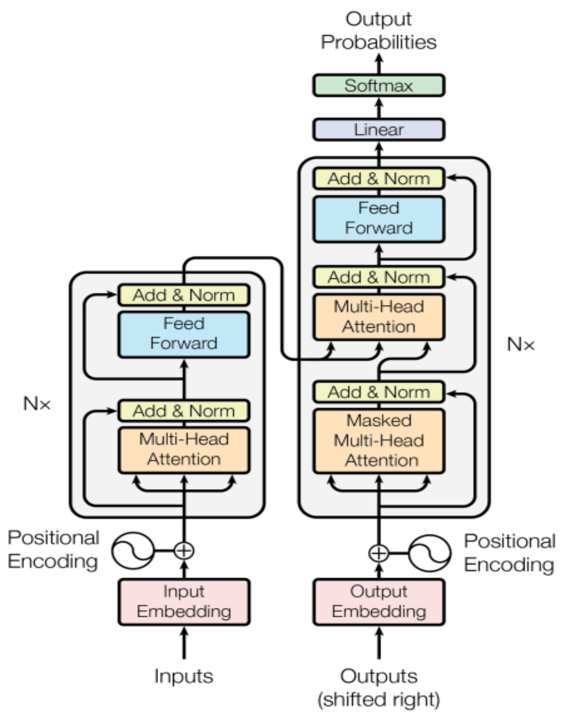

Figure 3: The original transformer architecture

Not long after the release of the original transformer, a transformer designed specifically for classification was designed. It was called a Vision Transformer(ViT) [3] figure shows the architecture of the vision transformer:

The vision transformer uses only the encoder part architecture of the original transformer. Before feeding the input image to the network it is preprocessed into patch embeddings. The vision transformer also uses positional embeddings instead of encoding. This is for the network to learn position of images regardless of orientation. After layers of transformer encoder, and MLP head is used to classify the

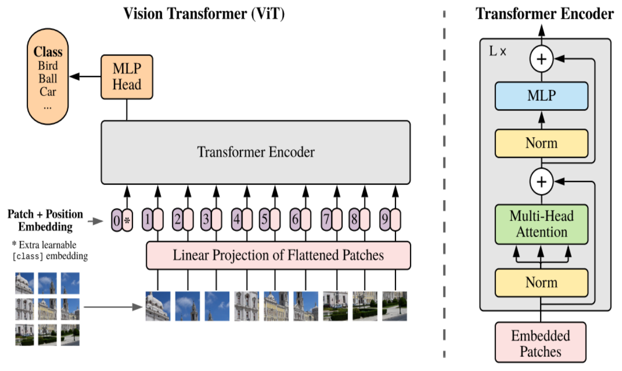

Figure 4: The architecture of the vision transformer

features learned by the transformer encoders, from there the image is classified.

This paper wants to explore using architecture similar to these two to predict responses of XOR Arbiter PUFs.

Related Works

Multiple studies with using deep learning models on PUFs are already done. Mursir et. al [1] in 2020, designed a machine learning model attacks on PUF. In their study they used Simulated and Silicon Challenge Response Pairs (CRPs) with PUFs designed using a Field Programmable Gate Arrays (FPGA) and some synthetic challenge response pairs using a simulator. In their study, they used multiple layers of fully connected layers that adjusts based on architecture of the PUF.

An almost similar study was done by Kumar and Niamat in 2018 [4], in which they also used FPGA to design ring oscillator PUFs. They investigated the performances of artificial neural network on different optimizers and different train-test splits.

One interesting study in 2020 by Yoon and Lee [5] called PUFGAN uses GANs to attack the PUF and restructure itself. In their study, they used vanilla GAN and Conditional GAN. They also used FPGA to design their PUFs for easier hardware reconfiguration. If the GAN is able to properly predict unrevealed CRPS it gives feedback to the hardware to reconfigure to resolve the exposed vulnerability. Figure 4 shows the working framework of PUFGAN.

One study by Nils Wisiol, et al[6] revisits neural network architectures of different papers and tried replicate their results. In this study, they have used both silicon CRPs designed form FPGA and their own open source simulator of CRPs called PYPUF. In which they questioned the reported accuracy of several papers as their results are not reproducible with low data complexity. This means that their results cannot be replicated with low number of CRPs for training as indicated in the refuted paper.

Finally, a study in 2022 by Sina Fard et. al[7] tried to use LSTMs and fully connected layers. They were able to replicate almost the same results as Mursir in 2020 using LSTM.

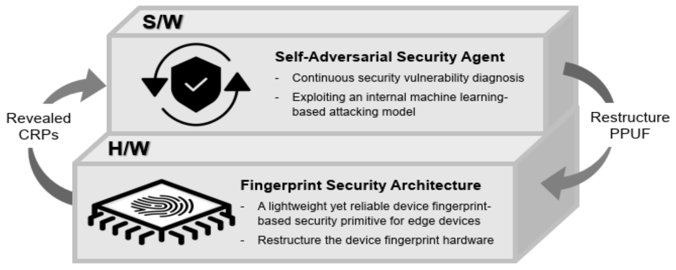
S/W
Self-Adversarial Security Agent
- Continuous security vulnerability diagnosis
- Exploiting an internal machine learning-based attacking model
H/W
Fingerprint Security Architecture
- A lightweight yet reliable device fingerprint-based security primitive for edge devices
- Restructure the device fingerprint hardware
Revealed CRPs
Restructure PPUF

Figure 5: Working framework of PUFGAN

With these recent studies, continuous study on attacks on PUFs are still relevant to further expose vulnerabilities related with PUFs to improve their design and develop counter measures. Study by Yoon and Lee [3] showcases both vulnerability exposure and countermeasure as they designed a GAN that is able to attack itself and feedback the hardware to update based on the security risks already learned by the deep learning model. This study will explore using Transformer Network to attack PUFs.

2. METHODOLOGY

Network Architecture

In this section we describe our proposed method using using transformer architecture to attack XOR Arbiter PUFs and its performance on different configurations. Figure 6 shows the model architecture for PUF Transformer used for attacking an Arbiter PUF.

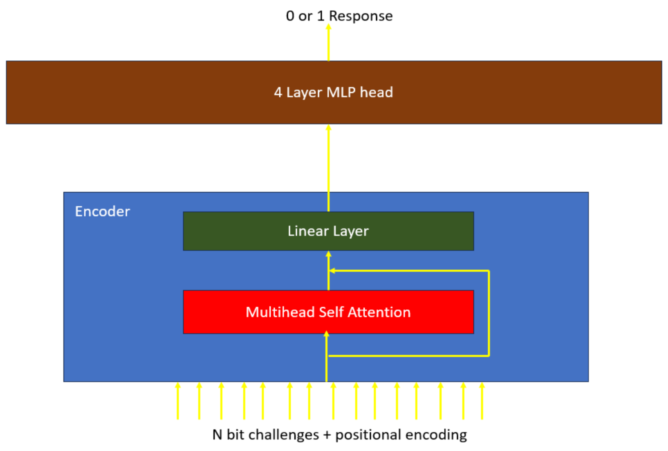

Figure 6: Architecture of PUF Transformer

Positional encoding is used to give information to the network regarding the order of the bits. Since the order of the input challenges are important in determining the response of the PUF. Equation 1 shows the positional encoding used for this paper:

$$\text{pe[pos]} = 10000 \sin\left(\frac{\text{pos}}{10000^2}\right) \tag{1}$$

The encoder layer is composed of one Multi-head Self Attention (MSA) layer and one linear layer. The MSA layer is used to the inputs with positional encoding to learn the dependencies of the input bits. Together with the positional encoding, the multi-head self attention layer should be able to learn and generalize from the input bits. The output of the MSA layer added to the original input is then fed to a linear layer with input size the same as the number of bits of the input challenge and output size of 8 times the input size.

The Multilayer perceptron (MLP) head then will process the output of the encoder layer. The 4 layer MLP head all have the same width of 8 times the size of the challenges. This is done to give way for learnable parameters that will be used for classification. Hyperbolic tangent activation function is used as activation for each of the hidden layer of the network

The final network designed has 8 heads for the transformer MSA layer. Dropout of 0.1 is used in the MLP layers for 32 bit challenges and dropout of 0.2 for 64 bit challenges to avoid overfitting and force the network to learn more complex features.

Table 1 shows the network's final parameter used for attacking different XOR Arbiter PUF configurations.

Table 1: Parameters for transformer attack method

`Library`	Pytorch
`Method`	Transformer architecture
`Architecture`	One Layer Encoder+MLP head
`HL Activation function`	tanh
`Output Activation function`	Sigmoid
`Loss Function`	BCE Loss
`Learning Rate`	0.005/0.001 with Cos Annealing
`CRP Source`	Synthetic
`Dropout`	0.1/0.2
`Number of heads`	8
`Batch Size`	10000

Experimental Set-up

We used CRPs from PYPUF[8] in our models for attacking XOR PUF. PYPUF is based on additive delay modeling used by Nils Wisiol [6] on their study to verify different machine learning models that attack PUFs. The model was tested on 2-XOR, 3-XOR, 4-XOR 32 bit and 2-XOR 64 bit configurations. The training loss used is Binary Cross Entropy using optimizer Adam. Cosine annealing is used to control the learning rate of the optimizer to be able to control the learning rate and able to move more conservatively the longer the training continues

For 32-bit, random inputs from 0 to 2^{32} are used as training dataset 1M are used for 2-XOR configuration while 4M are used for 3-XOR and 4-XOR configuration . This is done since 2^{32} total combinations are not possible to be tested. The goal is to train the network and still get high testing accuracy despite not seeing big percentage of challenge response pairs.

The network was also tested for 2-XOR 64 bit configuration and was tested with 4M training data and 9M training data.

Inputs used for training for this configuration are random inputs from 0 to 2^{60}.

For additional testing, another 1M, 10M, and 20M CRPs are generated and tested on the trained network to check the robustness of the accuracy of the trained network.

In the actual hacking scenario, these numbers of CRPs are required for the hacker to obtain either by stealing or other means in order to be able to use this model for hacking.

3. RESULTS AND DISCUSSION

Performance on 2 XOR-32 bit Arbiter PUF

A 2 XOR 32 bit Arbiter PUF is tested in the designed network. A 32 bit XOR PUF can have values from 0 to 2^{32} which is too big and impractical to be used as both for training and testing dataset. For this setting 2M random numbers from 0 to 2^{32} is generated using python and is converted to binary equivalents. These binary equivalents are then used to generate response from the simulated 2 XOR Arbiter PUF using PYPUF. Half of the generated CRPs are used as training data while the remaining half is used for testing. Figure 7 shows the learning curve for the 2-XOR Arbiter PUF.

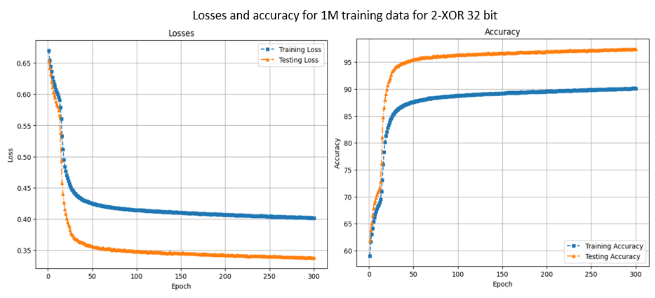

Figure 7: Learning curve for 2-XOR 32 bit training with 1M dataset

It can be observed from the figure that around epoch 20 the network already has a 90% accuracy in the test dataset. The network was able to generalize data efficiently on unseen data with only 1M training data. By the end of 300 epochs the network already have around 97.43% of testing data despite seeing only less than 1 % of the total available combinations in 32 bit challenges.

To verify the robustness of the accuracy of the model, the model was tested on different numbers of challenges and checked if it was still able to predict relatively close to the original test dataset. A set of 1M ,10M, and 20M challenges are generated and are fed to the network. Table 2 shows the accuracy for each set of challenges.

It can be observed from the table 2 that the accuracy of the trained network relatively have the same accuracy despite feeding it more unseen data. This proves that the network learned and generalized properly from 1M training dataset. To further check the performance of the network precision, recall, and F1 scores are calculated in table 3.

Table 2: Performance of the model on different number of test datasets

	True Pos	False Pos	True Neg	False Neg	Acc
1M	522542	14021	451949	11488	97.45%
10M	5226319	143074	4513968	116638	97.40%
20M	10447993	284658	9033825	233523	97.41%

Table 3: Performance of the model on classifier metrics

	Precision	Recall	F1 Score
1M	0.97387	0.97849	0.97617
10M	0.97335	0.97817	0.97576
20M	0.97348	0.97814	0.9758

It can be observed from the table 3 above that the model got almost perfect score on all precision, recall, and F1 Score. This means that the model performs extremely well on modeling the 2-XOR 32 bit Arbiter PUF.

Performance on 3 XOR-32 bit Arbiter PUF and 4 XOR-32 bit Arbiter PUF

The designed architecture was tested on 3-XOR 32 bit and 4-XOR 32 bit Arbiter PUF to check if the able will able to handle more chains in the XOR PUF configuration. During trials and experiments it was found that the network can handle these two configurations given ample amount of dataset. 4M dataset was used for training for 3-XOR and 4-XOR configuration. Figure 8 shows the learning curve for 3-XOR and 4-XOR configurations.

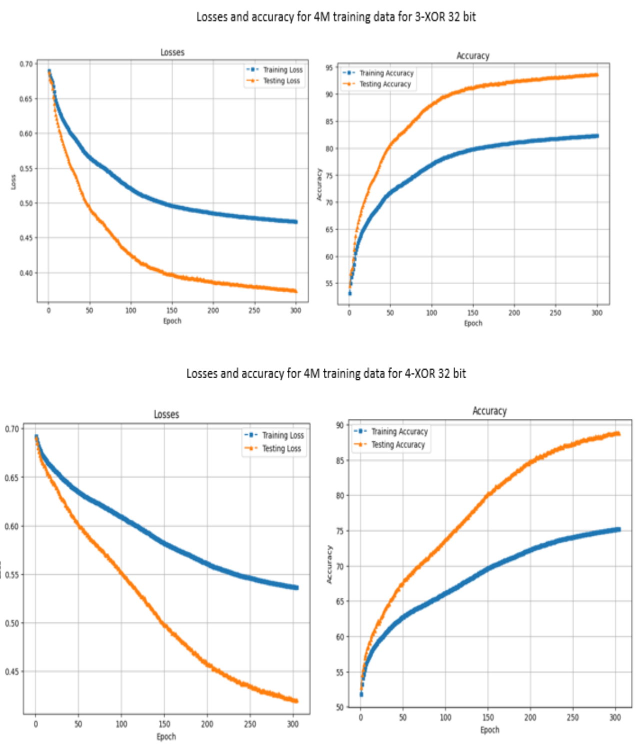

Figure 8: Learning curve for 3-XOR and 4-XOR 32 bit training with 4M dataset

It can be observed from the figure that with additional training data,3-XOR configuration was able to reach 93.72% accuracy. This is lower with 2-XOR training with only 1M training data. The network needs more data and epochs to handle the complexity of the 3-XOR configuration and perform much like with 2-XOR configuration. Compared to performance at 2-XOR, which reached 90% early at epoch 20, the increase in performance for 3-XOR is quite slow with it only reaching 90% at epoch 120. Nevertheless, it still good peformance considering the number of total CRPs for a 32 bit configuration.

On the other hand, it can be observed the 4-XOR only have 88.88% accuracy after 300 epochs. With 4 arbiter chains, the complexity is higher causing the network to experience difficulty reaching 90% even with 4M training dataset. Given more data and epoch the network would be able to hack the 4-XOR configuration with higher accuracy.

It can be observed from the learning curve for 4-XOR configuration is still linear despite reaching 300 epochs unlike for 3-XOR which already stabilizes towards the end of 300 epochs. This implies that given additional epochs, the network can still learn for 4-XOR configuration. Table 4 shows the performance of the model on 3-XOR and 4-XOR configurations while table 5 shows the precision, recall, and F1 score.

Table 4: Performance of the model on 3-XOR and 4-XOR configuration at 20M testing dataset

	True Pos	False Pos	True Neg	False Neg	Acc
k=3	9225999	772098	9227169	774732	93.69%
k=4	8641507	1111839	9110679	1135974	88.66%

Table 5: Performance of the model on classifier metrics with 3-XOR and 4-XOR configuration

	Precision	Recall	F1 Score
3-XOR	0.93642	0.93698	0.9367
4-XOR	0.8860	0.88382	0.88491

It can be observed from the table above that 3-XOR configuration still comparable performance to 2-XOR but 4-XOR has considerably lower scores but as discussed in the previous sections, additional epochs could still improve the performance for the 4-XOR configuration.

Performance on 2 XOR-64 bit Arbiter PUF

The performance of the designed network was tested on a 2-XOR 64 bit configuration to check if the network can learn on for longer N bit challenges. It has 2^{64} combinations of challenges that is considerably higher than 2^{32}. For initial testing, 5M random numbers between 0 to 2^{60} are generated and converted to binary as input challenges. The random number generator can only generate up to 2^{60} it should be enough to produce CRPs in which the model can learn. The dropout for 64 bit setting is increased to 0.2 as it is observed that the model overfits when used the dropout rate used is similar to the 32 bit configurations. The learning rate was decreased to 0.001 as it is observed during initial training that the network is having difficulty following a consistent loss decrease with 0.005 learning rate. Figure 9 shows the learning curve for 2-XOR 64 bit training with 4M of dataset.

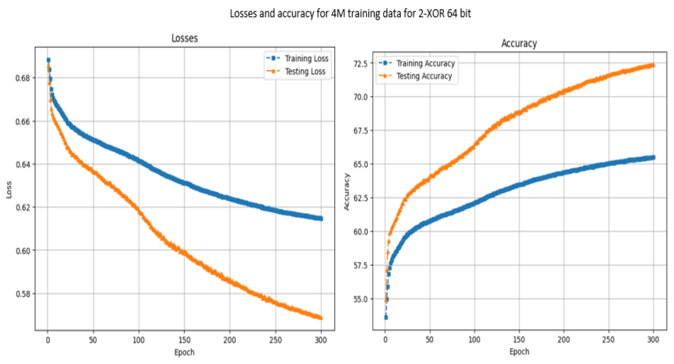

Figure 9: Learning curve for 2-XOR 64 bit training with 4M dataset

It can be observed that for 2-XOR 64 bit, the network struggles to learn with few data reaching only 72.40% at 300 epochs. This is expected as there are more combinations for a 64 bit combination but 72.40% performance that the network learns on this configuration. To check if additional training datasets will improve the performance of the network, training datasets were increased to 9M with testing dataset still of 1M. Figure 10 shows the learning curve when the training dataset is increased to 9M.

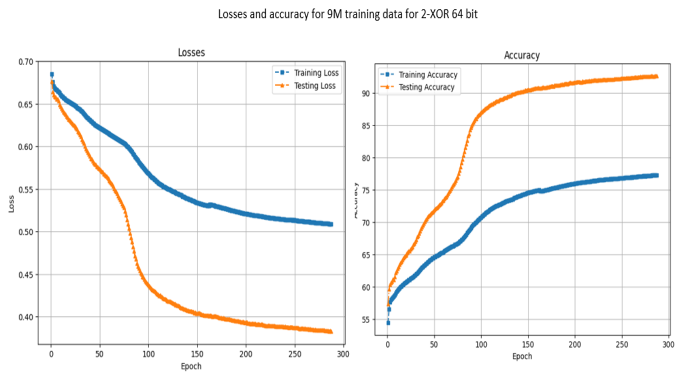

Figure 10: Learning curve for 2-XOR 64 bit training with 4M dataset

It can be observed from the learning curve that with 9M training dataset, the model performed better with accuracy increasing up to 92.66%. The model was able to reach 90 % accuracy at epoch 140. Table 6 shows the performance of the model at 4M and 9M of training while table 7 shows the precision, recall, and F1 score.

Table 6: Performance of the model on 2-XOR 64 bit configuration on varying datasets

	True Pos	False Pos	True Neg	False Neg	Acc
4M	7057998	2558606	7428426	2954969	72.43%
9M	9248380	703938	9283879	763802	92.66%

Table 7: Performance of the model on classifier metrics with 3-XOR on 4M and 9M training dataset

	Precision	Recall	F1 Score
4M	0.73394	0.70489	0.71912
9M	0.929278	0.92371	0.92648

4. CONCLUSION

In this paper, we have presented a machine learning based method for attacking 2 XOR arbiter PUFs. The attack has been accomplished by using transformer based neural network architecture. Using positional encoding, one multihead self attention layer, and three linear layers we are able to attack 2-XOR,3-XOR,4-XOR 32 bit configurations.

When attacking the 2-XOR 32 bit Arbiter PUF minimum of 1M challenges are enough to hack the PUF. The model was able to reach 97.5% accuracy for 300 epochs with 95% accuracy on the first 25 epochs.

The trained network was tested on several more unseen datasets to test the robustness and the generalization of the network. It was able to achieve almost equal accuracy even when exposed to both 10M and 20M unseen datasets. This shows that the network was able to model the 2-XOR 32 bit Arbiter PUF. Precision, recall, and F1 score of the network was computed and it was shown that the model was able to achieve almost perfect score on all categories.

The network was also tested on 3-XOR and 4-XOR 32 bit Arbiter PUFs and observed that the network is able to reach 93% accuracy for 3-XOR and 88% 4-XOR given 4M training datasets. The higher complexity of additional arbiter chains gives the necessity for more data needed for attacking or additional epochs.

Finally, the network also is also tested on 2-XOR 64 bit configuration. The performance of the network with 4M and 9M training datasets were observed and found that the network learns around 92.65% with only 200 epochs with 9M datasets while 4M dataset only learns up to 72% despite having 300 epochs. For longer input challenges more epochs and more datasets can improve the accuracy of the network.

This study opens the use of transformer based architecture for attacking XOR Arbiter PUFs. Future study may extend our work on attacking other XOR Arbiter PUF configurations or other PUF architecture. The transformer architecture can alse be improve to be able to learn complex configurations with less training datasets. There are currently few studies on attacking PUFs utilizing neural network for attacking. The characteristic of transformer to handle sequences using multihead self attention is beneficial for attacking different kind of PUFs.

5. REFERENCES

[1]K. T. Mursi, Bipana Thapaliya, Y. Zhuang, A. O. Aseeri, and Mohammed Saeed Alkatheiri, “A Fast Deep Learning Method for Security Vulnerability Study of XOR PUFs,” Electronics, vol. 9, no. 10, pp. 1715–1715, Oct. 2020, doi: https://doi.org/10.3390/electronics9101715.

[2]A. Vaswani et al., “Attention Is All You Need,” arXiv.org, Dec. 05, 2017. https://arxiv.org/abs/1706.03762

[3]D. Dosovitskiy et al., ”An image is worth 16x16 words: Transformers for image recognition at scale,” in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 3156-3164.

[4]S. Kumar and M. Niamat, “Machine Learning based Modeling Attacks on a Configurable PUF,” Jul. 2018, doi: https://doi.org/

[5]J. Yoon and H. Lee, “PUFGAN: Embracing a Self-Adversarial Agent for Building a Defensible Edge Security Architecture,” Jul. 2020, doi: https://doi.org/10.1109/infocom41043.2020.9155501.

[6]N. Wisiol, B. Thapaliya, K. T. Mursi, J.-P. Seifert, and Y. Zhuang, “Neural Network Modeling Attacks on Arbiter-PUF-Based Designs,” IEEE Transactions on Information Forensics and Security, vol. 17, pp. 2719–2731, 2022, doi: https://doi.org/10

[7]S. S. Fard, M. Kaveh, M. R. Mosavi, and S.-B. Ko, “An Efficient Modeling Attack for Breaking the Security of XOR-Arbiter PUFs by Using the Fully Connected and Long-Short Term Memory,” Microprocessors and Microsystems, p. 104667, Sep. 2022, doi: https://doi.org/10.1016/j.micpro.2022.104667.

[8]N. Wisiol et al., ”pypuf: Cryptanalysis of Physically Unclonable Functions,” Version v2, August 2021. [Online]. Available: https://doi.org/10.5281/zenodo.3901410