Effectiveness of Logistic Regression and Naïve Bayes Classification Algorithms on Arbiter PUFs

Dominic Lentini
Electrical and Computer Engineering
University of Florida
Gainesville, FL
dominic.lentini@ufl.edu

***Abstract*—**

Keywords—component, formatting, style, styling, insert (*key words****)***

I. Introduction

This paper demonstrates the ability to use machine learning algorithms to bypass Arbiter Physically Unclonable Functions (PUF). Arbiter PUFs utilize path delay by designing two paths, each with the same number of paths switching logic blocks that appear at the same time and are interconnected. These interconnected logic blocks will both receive the same input bit and use both paths as input. By repeating this logic and *x* times and comparing the output of both baths at the end of the circuit, the circuit takes *x* bits as an input which will determine the exact path the electrons will take. By designing these paths identically, the resulting bit will be determined entirely by process variation and other physical factors such as temperature [1]. Arbiter PUFs have a well-known vulnerability when it comes to being modeled using machine learning attacks due to the path delay model being a linear combination of the delay at each path at each path switching logic block. By expressing each logic block as having a delay value, the linear combinations of these delay values allow the two paths to be compared. To combat this, researchers have developed multiple ways to reduce the linearity of the delays, such as XOR-Arbiter PUFs which add more arbiter PUFs and uses an XOR comparison on the outputs of identically designed PUFs [2].

A large disadvantage to arbiter PUFs is increasing the number of Challenge Response Pairs (CRP) may be desirable to increase the reliability, however it also increases the vulnerability of machine learning attacks due to a smaller subset of CRPs being required to train the model. PUFs are inherently vulnerable to any sort of machine learning attack as the inability for the CRPs of a PUF to be 100% repeatable leads to a reduced accuracy requirement for an attack to be considered successful.

In this paper, two machine learning attacks are demonstrated on a 64-bit arbiter PUF. In the first attempt, a statistical approach is used to demonstrate the effectiveness of the PUF as it demonstrates the output of the circuit to be seemingly random. PUFs that are poorly designed would be able to be modeled easily using a purely statistical approach as the resulting bit could be predicted using hamming weight rather than a regressive classification. After the effectiveness of using a Naïve Bayes Classifier is shown to be poor, the Pypuf library [3] is used to create a logistic regression model of the PUF which results in a much higher accuracy of over 90% with very small subset of CRPs. A 64-bit challenge results in over 18 quintillion different CRP possibilities. During use of the PUF where a man-in-the-middle attack can implemented, a some number of CRPs could be captured and recorded. In this example, 12000 CRPs, a very small sample of what is possible, are used to train the model. The resulting technique is stress tested reducing the training size incrementally to find the minimum number of CRPs required to successfully model the PUF. The Naïve Bayes classifier is also retrained with each training and validation subset to show the effectiveness of a statistical approach with different CRP set sizes. We see from results that in a well designed PUF, a statistical approach to prediction should be very close to 50% as the distribution of the binary output is expected to be random.

II. Methodology

A Naïve Bayes Classifier with a Bernoulli distribution was used as a base line performance for a machine learning approach. This approach allows the data to be modeled using a purely statistical approach. This statistical approach would only work if the resulting bit and challenge bits were not random. While the CRPs are not random, their behavior, ideally is, making it hard to decern the resulting bit from the challenge. The classifier is implemented from the sklearn library. These results are compared to a Logistic Regression attack from the Pypuf library. The Bayes Classifier will take the challenge strings from the CRPs and determine a probability of the response bit by dividing the sum of the features by the length of the feature space. In this scenario, the hamming weight of the challenge is divided by the bit length of the challenge, 64. The algorithm was tested both with a prior bias of 0.5 and a uniform prior distribution. The prior probability of 50% yielded in a slightly better prediction. Due to the Bernoulli distribution classifier taking the hamming weight, it would incorrectly attribute the probability of the result being a 1 or 0 directly to the hamming weight as it does not take into account the physical path or path delays that are created by the challenge bits. This would not be the case with a feature space that is not fully binary. In this scenario however, the Bernoulli estimator works as a way to assess the PUFs response uniqueness, which attributes to the effectiveness of other algorithms. Below, the methodology for choosing whether or not to utilize the prior probability is shown.

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
[CV 1/5] END .....................fit_prior=True;, score=0.430 total time=   0.0s
[CV 2/5] END .....................fit_prior=True;, score=0.485 total time=   0.0s
[CV 3/5] END .....................fit_prior=True;, score=0.505 total time=   0.0s
[CV 4/5] END .....................fit_prior=True;, score=0.470 total time=   0.0s
[CV 5/5] END .....................fit_prior=True;, score=0.525 total time=   0.0s
[CV 1/5] END ....................fit_prior=False;, score=0.430 total time=   0.0s
[CV 2/5] END ....................fit_prior=False;, score=0.485 total time=   0.0s
[CV 3/5] END ....................fit_prior=False;, score=0.505 total time=   0.0s
[CV 4/5] END ....................fit_prior=False;, score=0.470 total time=   0.0s
[CV 5/5] END ....................fit_prior=False;, score=0.525 total time=   0.0s
```

```
GridSearchCV
estimator: BernoulliNB
BernoulliNB
```

```
print(grid.best_params_)
print(grid.best_score_)
```

```
{'fit_prior': 'True'}
0.483
```

The Pypuf library is unique in that it offers a method of feature extraction that accurately models an arbiter PUF

given CRP data [4]. Given the binary result bit, logistic regression is similar to a Naïve Bayes classifier in that it is able two classify samples into two different classes. However, unlike the Naïve Bayes classifier, logistic regression can linearly separate the data. The implementation within pypuf is an effective application of logistic regression because as shown in the trial, the pypuf application of the algorithm was very effective compared to the sklearn application of the algorithm. This is due to the extra manipulation of the data to linearly combine the challenge bits as a total path delay.

The set of 12000 was split into training and validation sets in several different ratios using the train_test_split function from the sklearn library. Sets were tested decreasing training size to determine the minimum requirements. The following training set sizes were used; 10000, 6000, 4000, 2000, 1000, 500, 400, 350. For each sized training set, both learning algorithms were used and trained on each set with a fresh kernel and 5-fold cross validation. The logistic regression attack was trained using a batch size of 1, a learning rate of 0.001, and 10 epochs. While the performance of the logistic regression is significantly better, the importance of performing a statistical attack such as the bayes naïve classifier should not be discounted. While there are metrics to determine the uniqueness and randomness of a PUF, the use of a statistical classifier proves the effectiveness of those metrics.

The pypuf library was developed to be used as a simulation. However it does feature an io function that allows real CRPs to be used as shown in the line of code below.

```
CRPTrain = pypuf.io.ChallengeResponseSet(XTrain, tTrain)
```

These CRP objects are very useful as they give you the ability to easily pull different attributes from the objects. One major flaw with using these objects however is they do require some non-obvious post processing to work properly. The responses for some reason get stored into the CRP object as an incompatible data type to be used with the attack functions. As shown below, the Logistic Regression task was able to train the model very quickly. Note that there was no cross validation with training the logistic regression model.

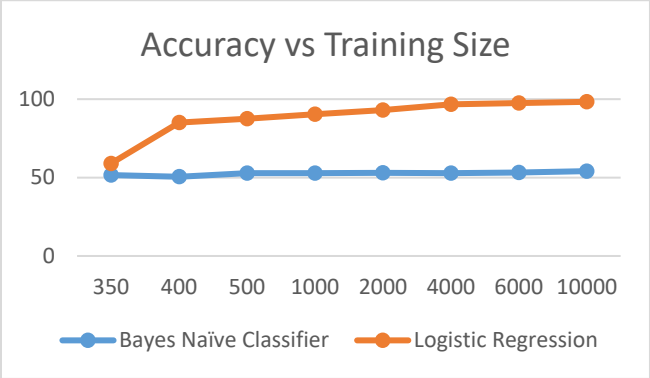
```
attack = pypuf.attack.LRAttack2021(CRPTrain, seed=3, k=1, bs=1, lr=.001, epochs=10)
model = attack.fit()

Epoch 1/10
990/990 [==============================] - 3s 3ms/step - loss: 0.6746 - accuracy: 0.5870 - val_loss: 0.3325 - val_accuracy:
0.9000
Epoch 2/10
990/990 [==============================] - 0s 407us/step - loss: 0.3488 - accuracy: 0.9150 - val_loss: 0.2045 - val_accurac
y: 1.0000
```

After training the models, the left over samples that are part of the validation set are run through the model to make predictions. These predictions are tested with the labels of the validation set to determine an accuracy score. For the Bayes Naïve classifier, the expected result should be 50% assuming the PUF is properly designed. For the logistic regression function, the ideal value would be 100%.

III. Results

As expected, the Bayes Naïve classifier performed at roughly a 60% effectiveness with the training data and 50% in test. This persisted with all ratios of the train-test split. The Logistic Regression classifier performed exceptionally well and was able to consistently remain above 80% on the test set with very few samples. The following graph shows the performance for each classifier with respect to the number of training samples.



From these results, we can conclude logistic regression is a very effective method for predicting the CRPs of an arbiter PUF. The Bayes Naïve classifier consistently yielded approximately 50% which shows the PUF is performing to expected quality throughout all the training sets. Logistic regression is continues to increase with more data and with only 500 samples, it is able to accurately predict the response to any challenge with 80% accuracy. However at 2000 samples, the accuracy is over 95% which is accurate enough to trick the designer or IP owner into thinking the device they are communicating is the PUF [5]. The 2000 samples required is also a very small number of CRPs given the 64-bit challenge length.

IV. Conclusion

With the effectiveness of machine learning attacks on PUFs being this high, it poses a strong security threat in the world of hardware due to the fact that very few CRPs are needed to train the model very accurately. This becomes a huge weakness as PUFs are intrinsically unreliable to some small extent. Therefore, the prediction model does not need to be 100% effective.

As these attacks become more prominent there are several directions to increase the security and reliability of PUFs. XOR arbiter PUFs, for example, are harder to predict due to the non-linear path delay from the XOR function between two arbiter PUFs. The pypuf library does however support XOR arbiter PUFs for several of the attack functions. Any current options that involve increasing the security of a PUF involves more money or more design time. An alternative to XOR PUFs, which can still be modeled, an alternative is the use of on-board encryption hardware that encrypts the CRPs at the source and destination. This would prevent the ability for a man-in-the-middle attack.

While this PUF was easily modeled, it is only one sample, for this experiment carry more weight, further investigation and repeatability for more PUFs, as well as different kinds of PUFs is needed.

References

[1] M. Bhargava, C. Cakir and Ken Mai, "Attack resistant sense amplifier based PUFs (SA-PUF) with deterministic and controllable reliability of PUF responses," 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2010, pp. 106-111, doi: 10.1109/HST.2010.5513106.

[2] Bhunia, S., & Tehranipoor, M. M. (2019). *Hardware security $Hswarup Bhunia, Mark M. Tehranipoor*. Morgan Kaufman Publishers.

[3] Wisiol, N. *et al.* (2021) *Cryptanalysis of physically unclonable functions¶, pypuf*. Available at: https://pypuf.readthedocs.io/en/latest/ (Accessed: December 4, 2022).

[4] Rührmair, U. et al. Modeling Attacks on Physical Unclonable Functions. in Proceedings of the 17th ACM Conference on Computer and Communications Security 237–249 (ACM, 2010). doi:10.1145/1866307.1866335

[5] Singh, Simranjeet & Bodapati, Srinivasu & Patkar, Sachin & Leupers, Rainer & Chattopadhyay, Anupam & Merchant, Farhad. (2022). PA-PUF: A Novel Priority Arbiter PUF. 10.48550/arXiv.2207.10526.