CS4150 LAB1 - PRNG AND PUF

Bas van't Spijker
1497944

Yuhang Tian
5219728

May 7, 2021

1 Pseudo Random Number Generator

1.1 PRNG Structure

Bit sequences are generated by an implementation of the AES algorithm (see figure 1). An initial key and initial message are both generated by a combination of a time function and a while loop.
First, the current time T_0 in nanoseconds is taken, after which an empty while loop is executed 10,000,000 times. This arbitrary large number introduces some desired noise, since the time it takes for a computer to execute this many empty while loops is inconsistent. After the while loops, the current time T_{+1} in nanoseconds is taken again, and the previous recorded time is subtracted ($T_{result} = T_{+1} - T_0$). This gives a large number, on which a modulo 255 operation is performed. The result is a single 8-bit byte B:

$$B = T_{result} \bmod 255.$$

The entire operation is performed 15 more times, and the resulting bytes are strung together in a single 128-bit sequence. One sequence is created for the initial key, and another one for the initial message.
These are then fed into the AES algorithm, which spits out the final 128-bit sequence. To generate more sequences after this point, the following operation is performed indefinitely:

1. The previously generated key becomes the new message.
2. The AES result becomes the new AES key.
3. Both are used to compute another 128 AES bit sequence.

For the NIST tests, ~25,600,000 bits were generated this way.

1.2 NIST Test & Van Neumann Processing

The generated bits were passed in 20 sequences of 10,000 for the first NIST tests, and in 20 sequences of 1,000,000 for the Random Excursions tests. The bits passed most of the Random Excursions tests, but failed the Binary Matrix Rank, Discrete Fourier Transform (FFT), Universal Statistical, and Serial tests. After van Neumann processing most tests now failed, except for the Binary Matrix Rank (now flipped), the Linear Complexity, and most of the Non-Overlapping Template tests. In this lab report we will discuss the Binary Matrix Rank and FFT test results.

1.2.1 Binary Matrix Rank

This test checks for "linear dependence among fixed length substrings of the original sequence". It divides a bitsequence into 32x32 matrices, and checks their rank. If the bits were truly random, the total count for each of the (three) ranks should follow a χ^2 distribution. The results of our tests show that the rank distribution is not random at all, as most of the matrices are rank R_{30}, this rank meaning that the matrix and its sub-matrices are all singular, non-invertible matrices. The very unbalanced occurrence of rank R_{30} matrices indicates severe linear dependency among the bits. This might have something to do with the fact that we use the output of the AES algorithm as the new key, and the old key as the new message. Using a pass through method like this could create some sort of linear pattern. After applying Van Neumann post-processing, 10/10 Rank tests now passed. This might be because the post-processing removes the linearity (probably) introduced by our pass-through method.

1.2.2 FFT

The Discrete Fourier Transform test checks for any (strong) periodicity in the bit sequence. In order to visualize why this test failed, we calculated the FFT graph ourselves, shown in figure 2. Here it can be seen that while almost all frequencies seem to be uniformly

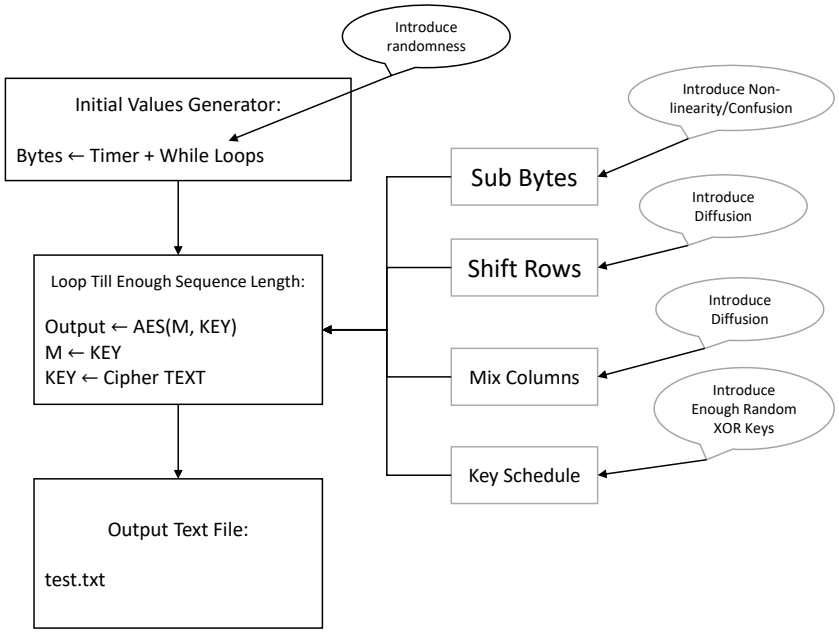
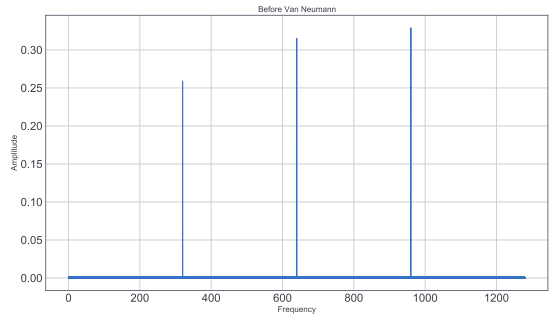
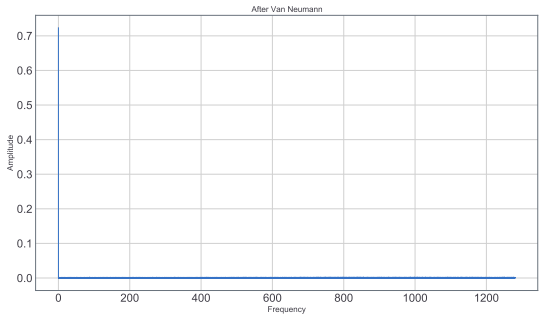


Figure 1: PRNG Structure

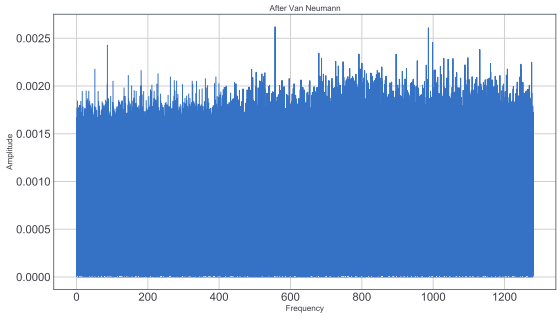
present, there are three extremely strong periodic patterns detected. We are unsure what exactly causes these periodic components. After applying the Van Neumann Processing (which also causes the loss of ~65% of all bits), these three frequencies are removed, but a new one at amplitude 0 is now present, see figure 2(b). This peak exists because the mean of the bits is no longer (close to) zero, i.e. there are significantly more one's than zero's or vice versa. Subtracting this mean from the bits to compensate for the imbalance results in an FFT graph that looks completely uniform, see figure 2(c). This shows that while the Van Neumann post-processing bit-array does not contain periodicity anymore, it still is not truly random either, as bits are disproportionately zeros or ones. For final comparisons sake, we have also included a FFT graph that was calculated from a list of equally many ones and zeros, shuffled by pythons *random.shuffle()* function.



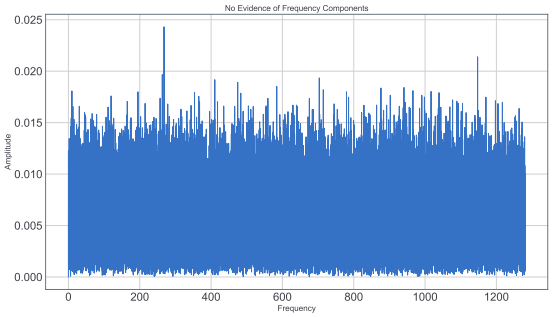
(a) Before Van Neumann Post Processing



(b) After Van Neumann Post Processing



(c) After Van Neumann Post Processing and Subtracted Mean



(d) Randomly shuffled list of equally many ones and zeros

Figure 2: Fast Fourier Transform of Outputs

2 Physical Unclonable Function

This section will analyze the functions of Arbiter-PUFs, XOR-PUFs and Interpose-PUFs, and apply two different ML models to attack them. The content has been divided into 5 sub-sections. It will first introduce the model settings relative to PUFs. Following that, it will explain which two ML algorithms will be selected to attack the PUF. Then, it will present the corresponding attack results of Arbiter-PUF, XOR-PUF and Interpose-PUF, respectively.

2.1 Model Description

Fig 3 shows the structure of an arbiter PUF. An n-bit arbiter PUF consists of n stages, every stage being formed by a pair of multiplexers. Each stage will add some $\pm$ delay between upper branch and lower branch signals. Since the delay of each stage is an intrinsic property determined by manufacturing variance, it is theoretically infeasible to clone another identical n-bit APUF. The inputs for an arbiter PUF are composed of a set of individual challenging bits $\{c_0, c_1, c_2..., c_{n-1}, c_n\}$, each of which is also a selecting bit for a pair of multiplexers. A challenging bit determines whether the input signal should be crossed. In this figure, bit 0 for a stage represents not crossing where the higher multiplexer selects the upper branch signal and lower one does the opposite; in contrast, bit 1 for a stage means crossing the signals. The crossing- or not crossing decides the delay/variance to be added between the upper signal and lower signal. The upper signals and lower signals propagate through the n stages and finally reach the arbiter (the arbiter also causes delay) which will decide the output response according to which signal arrives first. For identical challenging-bit sets, the arbiter should, with high probability, give the same response (sometimes the output might be influenced by the noise or device ageing).

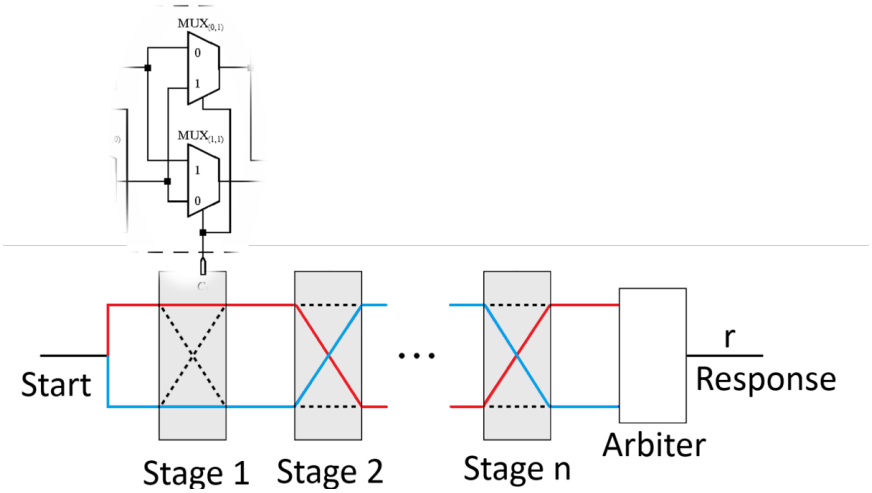


Figure 3: Structure of APUF

According to the above description, we can see that there is no "direct linkage" - linearity or kinds of explicit relationships - between each challenging bit and the final response, as each challenging bit acts autonomously, only influencing its current-stage upper and lower signals. Therefore, when considering how a stage challenging bit influences the arbiter's response, the consecutive stages after that stage should also be taken into consideration. Eq. 1 shows how it works. In the equation, k is the total number of stages, i is the current stage, c is the stage challenge bit, ϕ is the stage feature. The stage feature can be defined as they way that a challenging bit affects the result observed from the arbiter's side. ϕ_{i+1} is the feature of arbiter. These equations transform a set of challenging bits into a training feature vector.

$$\phi_i = \prod_{l=i}^{k} (-1)^{c_l} \text{ For } 1 \leq i \leq k \tag{1}$$

$$\phi_{k+1} = 1 \tag{2}$$

To make the model simpler, it is assumed that the delay for each stage is an undefined constant, so the set of delays becomes $\{w_1, w_2, w_3..., w_k, w_{k+1}\}$ where w_{k+1} is the delay of arbiter. Therefore, the total delay at the end can be calculated by using Eq 3 the decision of the arbiter can be determined by Eq. 4.

$$\Delta D = \vec{w}\vec{\phi}^T \tag{3}$$

$$r = \begin{cases} 1 & \text{If } \Delta D \leq 0 \\ 0 & \text{If } \Delta D > 0 \end{cases} \tag{4}$$

2.2 ML Algorithms

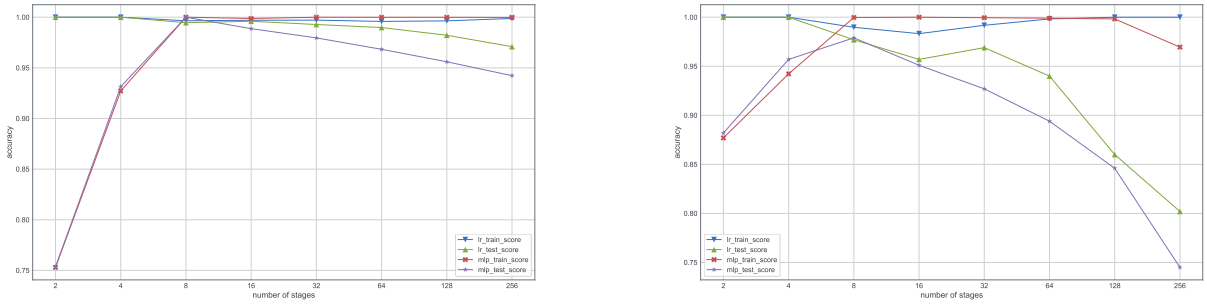
For the selection of algorithms, we chose the first one as the recommended and provided one - *logistic regression*, and for the other one we selected *multi-layer perceptron* since it is a favourable model to deal with the non-linearity (linear inseparable problems). More details of the latter model will be presented in the xor-PUF section. The algorithms are implemented using *Scikit-learn*.

2.3 Arbiter PUF

Arbiter PUF, or a-PUF for short, is a fundamental type of PUF family. The function of an a-PUF is the simplest one compared to the other two types of PUFs. There are no designing parameters except for the number of stages, so this section will explain the design of the experiments, with the goal of exploring how this parameter provides the defence against ML algorithm attacks.

The ML model hyper-parameters are:

- Logistic Regression: solver="lbfgs"
- Multi-layer Perceptron:
  1. hidden_layer_sizes=(n_stage, 2)
  2. activation='relu',
  3. learning_rate='adaptive',
  4. learning_rate_init=0.01



(a) Challenge-Response Pairs = 5000

(b) Challenge-Response Pairs = 500

Figure 4: A-PUF

In Fig 4, we fixate the challenge-response pairs each time and increase the number of stages exponentially based on 2 until $2^8 = 256$. On the left-hand side, we have 5000 samples, while on the right-hand side, we only have 500 samples. The repetition number of the experiments is 10 (epoch=10).
Logistic Regression always performs well, achieving more than 95% accuracy when predicting, so its results prove that the feature vector that was converted from challenge bits has a linear relationship with the label of response. In general, if the input and output have a linear relationship, LR tends to fit quite well, even when the number of the training samples is limited.
For the Multi-Layer Perception, the number of parameters in the hidden layers that need to be trained is proportional to the number of stages. According to Fig. 4(a), if the number of training samples is large enough, the results can also be good, achieving almost the same performance as LR. It can be observed that the accuracy of MLP does not exceed 90% when the number of stages is 2, because the number of parameters of MLP is proportional to the number of stages - model setting - and the parameters are too few to support the accurate prediction. However, if there are not enough training samples, the performance of MLP is insufficient, and it even becomes worse if the number of stages increases.

In conclusion, an a-PUF has a disadvantage of linearity - the feature vector and response bit obey a linear model. This defect makes attacking using linear ML algorithm such as logistic regression classifier possible. Even with a limited number of training samples, logistic regression can still effectively attacks an a-PUF easily. Adding more stages cannot improve things because LR is robust against the increasing number of stages. In order to avoid this situation, it is necessary to introduce non-linearity to an a-PUF.

2.4 XOR PUF

An xor-PUF combines two or more a-PUFs, the result being drawn from the xor calculation of all a-PUFs, bringing non-linearity to the PUF, since the results of the xor operations are linearly inseparable as shown in Fig. 5. Unlike other 2-bit calculations like and, not and or, which can be separated by a single line in a 2-dimensional space, xor cannot be classified. To overcome the non-linearity, we decide to use

Bit 0	Bit 1	Result
0	0	0
0	1	1
1	0	1
1	1	0

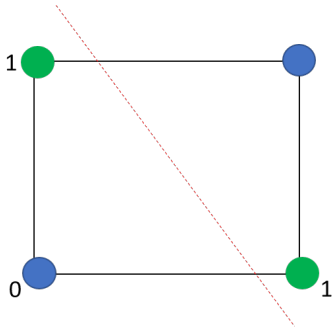


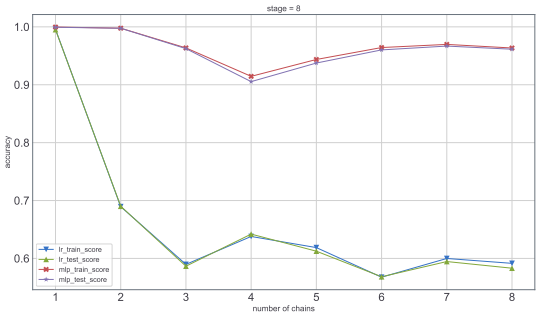
Figure 5: XOR Non-linearity

MLP. MLP uses "relu" as an activation function that can learn and fit the non-linearity of the model. Compared to a-PUFs, xor-PUFs have one more parameter which is the number of chains.

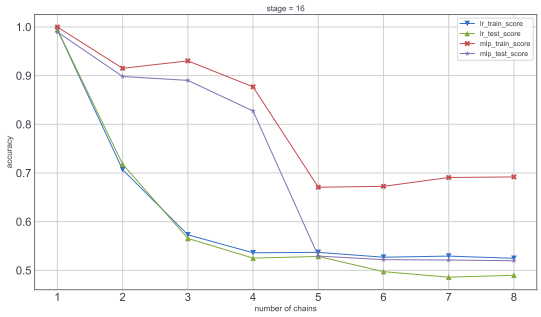
The ML model hyper-parameters are:

- Logistic Regression: solver="lbfgs"
- Multi-layer Perceptron:
  1. hidden_layer_sizes=(n_stage, n_chain)
  2. activation='relu',
  3. learning_rate='adaptive',
  4. learning_rate_init=0.01

The only thing that is changed compared to the previous model is the hidden layer hyper-parameter. We set the node's number of the first hidden layer equal to the number of stages and the node's number of the second hidden layer equal to the number of chains. The reason for this setting is that we expect the parameters in the first hidden layer to mimic the decisions of all autonomous stages and the parameters in the second hidden layer to mimic the decisions of all arbiters at the end of each chain. The "relu" activation function is for breaking the linearity of learning and adding the ability to perform the non-linear learning and decision.



(a) Number of Stages = 8



(b) Number of Stages = 16

Figure 6: XOR-PUF

In Fig. 6, we fixate the number of stages each time and gradually increase the number of chains in the model to observe the performances of two different ML classifiers. The repetition number of the experiments of the left-hand side model is 10 whereas the repetition number of the experiments of the right-hand side model is 5 (it really consumes lots of time for the code running, so we reduce the repetition). The sample's number - the number of challenge-response pairs - is 5000 which is adequate for the stage number = 8 ($5000 > 2^8$) but not quite enough for the stage number = 16 ($5000 < 2^{16}$). In overview, when there is only one chain, xor-PUF degenerates to an a-PUF, so both classifiers achieve a high predicting accuracy; however, when the chain's number is equal or larger than 2, MLP always performs better than LR.

LR does not have resistance to non-linearity. When two a-PUFs combine into an xor-PUF, its test accuracy is almost like a random guess - around 50%. It proves that xor-PUF has the ability to defend against linear ML model attacks to some extent.
For MLP, when the number of samples is large enough to support training its parameters, it can effortlessly achieve very high prediction accuracy, more than 90%. Nevertheless, when the number of chains grows larger, it becomes incapable. We can also compare figure 6(b) with figure 4(a) which shows that a 16-stage xor-PUF is much better than a 64-stage a-PUF.

In conclusion, xor-PUF has the resistance to defend linear ML algorithm attacks, but it can still be predicted by some powerful non-linear ML algorithms such as MLP. MLP only works fine when there are enough training samples to support, so an attacker needs to obtain larger numbers of challenge-response pairs before implementing an attack. Combing a-PUFs in a parallel xor way can achieve higher security than combining in a serial way.

2.5 Interpose PUF

An interpose-PUF, or i-PUF for short, consists of two xor-PUFs. One of the xor-PUFs will get a new stage inserted in its original structure, the challenging bit of which is the result of another xor-PUF, as shown in Fig. 7. As mentioned, there is no linear relationship between a challenging bit and the final arbiter's output, so this measurement introduces another way to add non-linearity to PUFs. There will be two feature vectors for i-PUF, the upper feature vector and the lower feature vector. The only difference is that the lower feature vector includes an interposed challenging bit from the upper xor-PUF, the content and the position of which are not disclosed to the attacker. Therefore, for the model training, we can only use the upper feature vector, otherwise, the insertion becomes meaningless. An i-PUF owns four designing parameters - the number of stages, the number of chains in the upper layer, the number of chains in the lower layer and the position of the interposed bit.

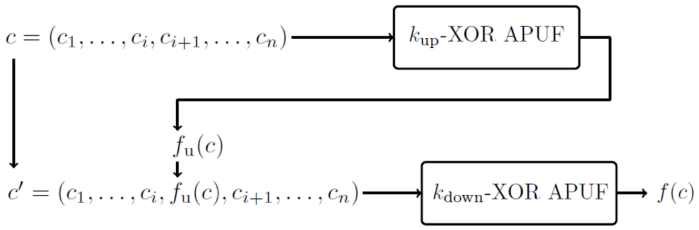


Figure 7: I-PUF Structure

The ML model hyper-parameters are:

- Logistic Regression: solver="lbfgs"
- Multi-layer Perceptron:
  1. hidden_layer_sizes=(n_up_stage, n_up_chain + n_down_chain)
  2. activation='relu',
  3. learning_rate='adaptive',
  4. learning_rate_init=0.01

The settings of the two classifiers are similar to the previous one except for the hidden layer hyper-parameter of MLP, where the number of nodes in the first hidden layer is the stage's number of upper xor-PUF and the number of nodes in the second hidden layer is the sum of chain's number of those two xor-PUFs. We expect that the first hidden layer acquires knowledge of how each stage works - the delay of each stage - and the second hidden layer learns how the results are combined.

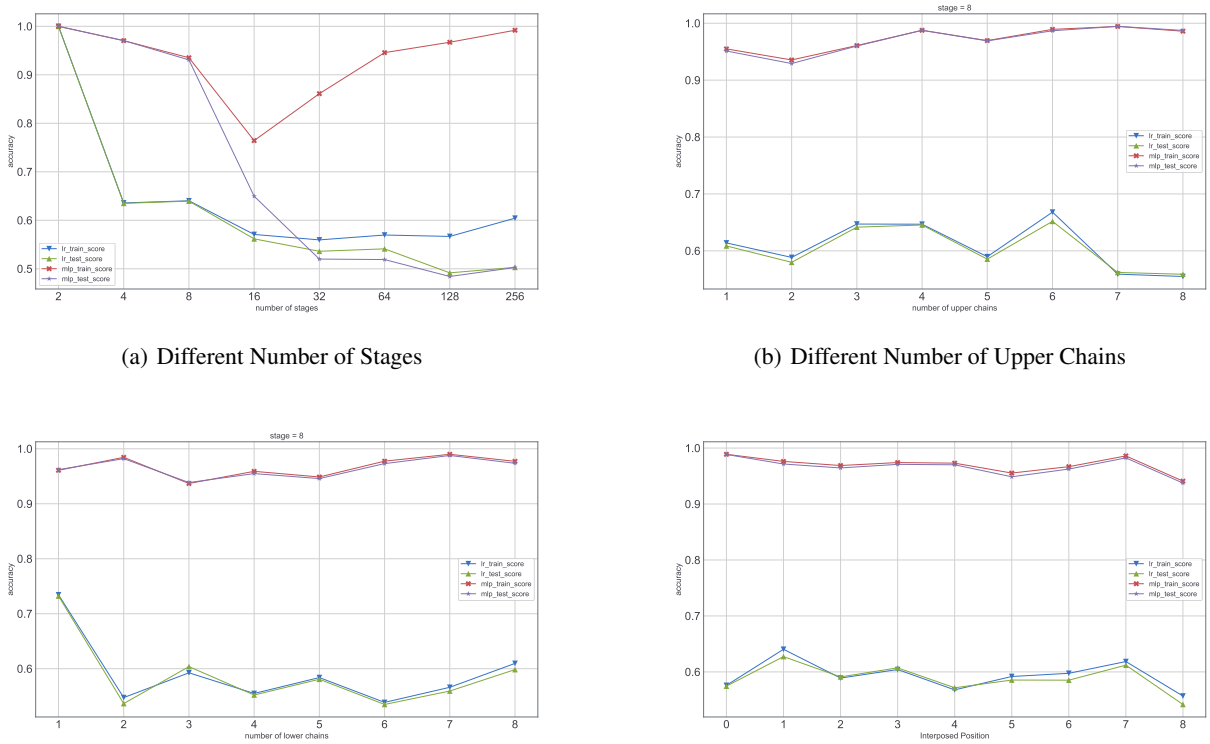
In Fig. 8, we examine the four designing parameters of an i-PUF separately. Fig. 8(a) shows the result of the accuracy of two classifiers when the number of stages increases exponentially based on 2 till $2^8 = 256$. Fig. 8(b) and Fig. 8(c) reveal how accuracy changes when gradually increasing the number of chains in the upper xor-PUF and lower xor-PUF respectively. Fig. 8(d) show how different positions of interposed challenging bit affect the accuracy. The four figures will be individually and sequentially addressed in the next section. Assuming that attackers can only obtain the constant and limited number of challenge-and-response pairs for training their models, 5000 samples are the global setting here.

2.5.1 Stage Number

To analyze the independent effect of the number of stages on the ML model for i-PUF attacks, the other three designing parameters are kept constant. The repetition number of the experiments is 10.

- the number of upper chains: 3
- the number of lower chains: 3
- the position of interposed bit: in the middle

According to figure 8(a), with a growing number of stages, the testing accuracy of the LR classifier dramatically decreases. Besides two-stage i-PUF, LR is powerless for any other i-PUFs which stage's number is larger than 2, and it even falls into random guess accuracy after the number of stages hits 32. Therefore, i-PUF definitely has a reliable and high resistance to linear ML model attacks. Turning to MLP classifiers, before the number of stages becomes equal to 16, its testing accuracy approximates its training accuracy, usually exceeding 90% - a sufficiently high number. However, a strange deviation occurs when the number of stages larger than 16. Although



(a) Different Number of Stages

(b) Different Number of Upper Chains

(c) Different Number of Lower Chains

(d) Different Interposed Positions

Figure 8: Parameters in I-PUF

the training score is pretty high (almost more than 90%), the testing accuracy is far too low. We suspect that the phenomenon is caused by *overfitting*, due to the limited training samples fed in. To some extent, it proves that MLP has the potential capability to achieve sufficiently high prediction accuracy if the attacker obtains ample samples and owns a powerful computational machine. Therefore, i-PUF is still risky in front of non-linear ML models.

2.5.2 Chain Number

There are two kinds of chain numbers, the upper xor-PUF's chain number and the lower xor-PUF's one. To analyze the independent effect of one of the chain's numbers on the ML model for i-PUF attacks, the other three designing parameters are maintained constant. The repetition number of the experiments is 10.

- the number of stages: 8
- the number of chains of another xor-PUF: 3
- the position of interposed bit: 4

According to figure 8(b) and figure 8(c), obviously, MLP can effectively predict the outputs for i-PUFs. Due to an adequate number of training samples ($5000 > 2^8$), the hidden layers of MLP comprehensively acquire the knowledge of delays and non-linearity combinations, which points out that i-PUF still cannot fully resist non-linear ML attacks. The reason for using "not fully" here is that if the attacker's number of training samples is not high enough, the parameters in MLP cannot fully be trained as figure 8(a) shows. There is one more notable observation to be made about figure 8(c) When the number of chains from the lower xor-PUF equals 1, LR can still reach a 75% prediction accuracy, but adding one chain more can significantly reduces the accuracy, bringing LR down to a random guess model. Compared with it, in figure 8(b), the accuracy of LR levels are mostly low with a few fluctuations. This indicates that when designing an i-PUF, the number of chains in lower xor-PUF is slightly more important than the number of chains in upper xor-PUF. For example, if only one of them can be strengthened, the one from lower xor-PUF should be considered first.

2.5.3 Interposed Positions

To analyze the independent effect of the different interposed positions on the ML model for i-PUF attacks, the other three designing parameters are maintained constant. The repetition number of the experiments is 10.

- the number of stages: 8
- the number of upper chains: 3
- the number of lower chains: 3

Figure 8(d) shows that different interposed positions cannot adversely influence the prediction results. MLP still solves the puzzle no matter where the result upper xor-PUF is inserted.

2.5.4 Question

What will be the max possible accuracy when attacking the i-PUF with machine learning as in previous assignments? Why?

To discuss the highest accuracy that the attacker can achieve, we assume that she has the ability to obtain an adequate number of samples to train her ML model which has the capability to deal with the non-linearity, and she has a powerful computer to process the data. In this situation, due to the delay of each stage being constant, we can say that the prediction can achieve an accuracy of more than 90%. However, if noise exists in the i-PUF device which sometimes makes the outputs for identical inputs vary, the attacking accuracy will be (slightly) lower. Let's say that the precision of a given i-PUF is merely 90% - then the outputs are identical for the corresponding inputs 90% of the time, so the attacking accuracy could hardly exceed that number.