CS771 : Introduction to Machine Learning
Assignment - 1
Team: StakeInsight

Dhananjay Dixit
220351
Dept. of Aerospace Engineering
IIT Kanpur
dhananjayd2222@iitk.ac.in

Sahil Kumar
22936
Dept. of Mechanical Engineering
IIT Kanpur
sahilkur22@iitk.ac.in

Pratham Garg
220808
Department of Electrical Engineering
IIT Kanpur
prathamg22@iitk.ac.in

Ritesh Hans
220893
Dept. of Computer Science and Engineering
IIT Kanpur
riteshhans22@iitk.ac.in

Aryan Gautam
220222
Dept. of Aerospace Engineering
IIT Kanpur
garyan22@iitk.ac.in

Abstract

This is our solution to the first assignment of the CS771 course. We aim to demonstrate, through mathematical derivation, the existence of linear models capable of perfectly predicting the responses of a Cross-Connection PUF (COCO-PUF). Moreover, we will show that these linear models can be accurately estimated with a sufficient number of challenge-response pairs.

Preprint. Under review.

1 Part 1

1.1 Arbiter PUFs – A Brief Introduction

Arbiter Physically Unclonable Functions are hardware systems that capitalise on unpredictable differences in data transmission speed in different iterations of the same design to implement security.

They consist of a series of multiplexers (mux'es, hereafter) each having a selection bit. A particular set of selection bits is said to form a "question/challenge", and depending on these selection bits, the signal that reaches the end of the PUF first is said to be the "answer". The answer to a particular challenge is therefore unique to the hardware of that particular PUF.

1.2 Using ML to crack Arbiter PUFs

It has been found that knowing responses to a relatively small number of challenges, a model can be trained to predict answers to any other challenge for a particular arbiter PUF, the analysis for which is shown below:

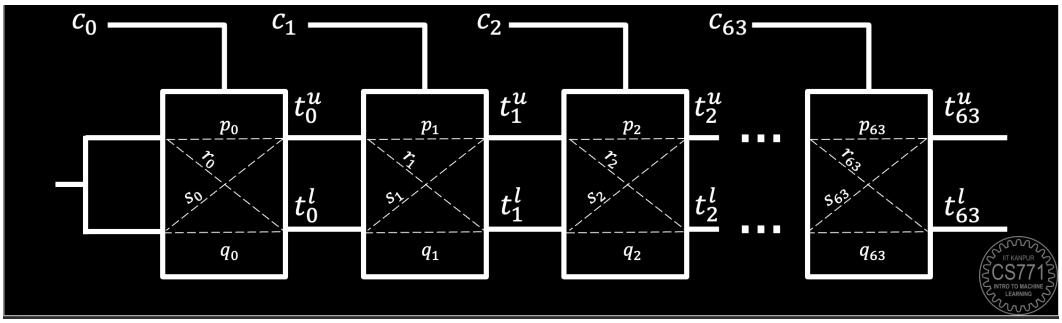

Figure 1: Analysis – Arbiter PUF

t_i^u is the (unknown) time at which the upper signal leaves the i-th mux.
t_i^l is the time at which the lower signal leaves the i-th mux.
All muxes are different so that p_i, r_i, s_i and q_i are distinct.
Therefore, the answer is 0 if $t_{31}^u < t_{31}^l$ and 1 otherwise.
Now,

$$t_1^u = (1 - c_1).(t_0^u + p_1) + c_1.(t_0^l + s_1)$$
$$t_1^l = (1 - c_1).(t_0^l + q_1) + c_1.(t_0^u + r_1)$$
$$\Delta_i = t_i^u - t_i^l$$

$$\begin{aligned}\Delta_1 &= (1 - c_1) \cdot (t_0^u + p_1 - t_0^l - q_1) + c_1 \cdot (t_0^l + s_1 - t_0^u - r_1) \\ &= (1 - c_1) \cdot (\Delta_0 + p_1 - q_1) + c_1 \cdot (-\Delta_0 + s_1 - r_1) \\ &= (1 - 2c_1) \cdot \Delta_0 + (q_1 - p_1 + s_1 - r_1) \cdot c_1 + (p_1 - q_1)\end{aligned}$$

$$\text{Let } d_i = (1 - 2c_i)$$

$$\Delta_1 = \Delta_0 \cdot d_1 + \alpha_1 \cdot d_1 + \beta_1$$
$$\alpha_1 = \frac{p_1 - q_1 + r_1 - s_1}{2}, \beta_1 = \frac{p_1 - q_1 - r_1 + s_1}{2}$$

A similar relation holds for any stage i:

$$\Delta_i = d_i \cdot \Delta_{i-1} + \alpha_i \cdot d_i + \beta_i$$

Let $\Delta_{-1} = 0$ (absorb initial delays into p_0, q_0, r_0, s_0)
We can keep going recursively:

$$\Delta_0 = \alpha_0 \cdot d_0 + \beta_0 \quad (\text{since } \Delta_{-1} = 0)$$
$$\Delta_1 = \Delta_0 \cdot d_1 + \alpha_1 \cdot d_1 + \beta_1$$

plugging in the value of Δ_0, we get

$$\Delta_1 = (\alpha_0 \cdot d_1 \cdot d_0) + (\alpha_1 + \beta_0) \cdot d_1 + \beta_1$$
$$\Delta_2 = \alpha_0 \cdot d_2 \cdot d_1 \cdot d_0 + (\alpha_1 + \beta_0) \cdot d_2 \cdot d_1 + (\alpha_2 + \beta_1) \cdot d_2 + \beta_2$$

noticing the pattern,

$$\Delta_{31} = w_0 \cdot x_0 + w_1 \cdot x_1 + \ldots + w_{31} \cdot x_{31} + \beta_{31} = \mathbf{w}^\top \mathbf{x} + b$$

where

$$x_i = d_i \cdot d_{i+1} \cdot \ldots \cdot d_{31}$$
$$w_0 = \alpha_0$$
$$w_i = \alpha_i + \beta_{i-1} \quad \text{for } i > 0$$

This gives us the final difference in timings experienced between both the signals for a PUF, Δ_{31} as an affine function, $\mathbf{w}^\top \mathbf{x} + b$

1.3 Using ML to Cross-Connection PUF (COCO-PUF)

A COCO-PUF uses 2 arbiter PUFs – say **PUF 0 and PUF 1**, each of the arbiter PUFs has its own set of multiplexers with different delays.
Given a challenge, it is fed into both ,PUF 0 and PUF 1. The responses are generated such that the lower signal from PUF 0 compete with the lower signal from PUF 1 using an arbiter called **Arbiter0**, and the upper signal from PUF 0 competes with the upper signal from PUF 1 using an arbiter called **Arbiter1**. The former is treated as **Response0** and the later as **Response1**.
If the signal from PUF0 reaches first, Response1 is 0 else if the signal from PUF1 reaches first, Response1 is 1.

So in this section, let us show that a COCO-PUF can be cracked using a Linear ML model. According to the question, the COCO-PUF which we have been given is a COCO-PUF with 32 bit challenges. So we will show our analysis on a 32 bit COCO-PUF. Analysis for COCO-PUFs with higher bits of challenges follow similarly to the analysis below.

In the previous section we showed that delay after any stage i for an arbiter PUF is given by :

$$\Delta_i = d_i \cdot \Delta_{i-1} + \alpha_i \cdot d_i + \beta_i$$

where

$$d_i = (1 - 2c_i), \alpha_i = \frac{p_i - q_i + r_i - s_i}{2}, \beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$

Since

$$\Delta\_1 = 0$$

, we observe the following :

$$c_1 \cdot \Delta_1 = c_1 \cdot (\alpha_0 \cdot d_0 + \beta_0)$$
$$c_2 \cdot \Delta_1 = c_2 \cdot (\alpha_0 \cdot d_1 \cdot d_0 + (\beta_0 + \alpha_1) \cdot d_1 + \beta_1)$$
$$c_3 \cdot \Delta_2 = c_3 \cdot (\alpha_0 \cdot d_2 \cdot d_1 \cdot d_0 + (\beta_0 + \alpha_1) \cdot d_1 \cdot d_2 + (\alpha_2 + \beta_1) \cdot d_2 + \beta_2)$$

$$\vdots$$

Adding all till Δ gives us the recurrence generalization Δ_{31}, the last output.

$$c_{32} \cdot \Delta_{31} = c_1 \cdot w_0 \cdot x_0 + c_2 \cdot w_1 \cdot x_1 + \cdots + c_{32} \cdot w_{31} \cdot x_{31} + c_{32} \cdot \beta_{31} = \sum_{i=1}^{n=32} c_i \cdot \Delta_{i-1}$$

$$\sum_{i=1}^{n=32} c_i \cdot \Delta_{i-1} = \sum_{i=1}^{n=32} w_i \cdot x_i + \sum_{j=i+1}^{n=31} c_j + \sum_{i=1}^{n=32} c_i \cdot (s_i - p_i)$$

Hence the final expression for the Upper Signal reaching time is

$$t_n^u = \sum_{i=1}^{n=32} w_i \cdot x_i + \sum_{j=i+1}^{n=32} c_j + \sum_{i=1}^{n=32} c_i \cdot (s_i - p_i - \beta_{i-1}) + \sum_{i=1}^{n=32} p_i$$

where $\mathbf{w}$, $\mathbf{x}$ are 32 dimensional vectors and $\mathbf{W}$, $\mathbf{X}$ are 33 dimensional, just including the bias term $W_{32} = \beta_{31}$ and $X_{32} = 1$. Each term of $\mathbf{w}$, $\mathbf{x}$ being:

$$b = \beta_{31} \quad w_0 = a_0 \quad w_i = \alpha_i + \beta_{i-1} \quad x_i = \prod_{j=1}^{31} d_i$$

Final feature Vector for the Linear Model is -

$$\mathbf{X} = [c_1, c_2, \cdots, c_n, x_1 \cdot \sum_{i=2}^{n=32} c_i \cdot x_2 \cdot \sum_{j=3}^{n=32} c_i \cdot \cdots\cdots\cdots, x_{31} \cdot c_{32}]$$

Remark: The only parameter we need for predicting the output of Δ_{31}, given a new challenge, is the vector $\mathbf{W}$, which we extract from pre-existing efficient linear models which use the feature vector $\mathbf{X}$ for multiple challenge-response pairs as available data for training.

2 Dimensionality of our new feature map

As mentioned above the feature vector obtained contains 63 terms hence the dimensionality of our Map is **63**.

3 Solution to Q3

Now we have obtained the upper signal of the COCO as a Linear Model it can be used to create a classifier model which can correctly predict the Response 1 and Response 0 generated by COCO PUF.

So now, after analyzing and deducing $\mathbf{w}$, $\mathbf{x}$, b for the upper signal time t_n^u the COCO-PUF problem, let the Upper signal reaching time be defined as:

$$\begin{aligned}\Delta_{31}^u = {} & b_1 + w_0 c_0 + w_1 c_1 + w_2 c_2 + \ldots + w_{31} c_{31} \\ & + w_{32}\left((1-2c_2)\ldots(1-2c_{31})\right) \\ & + w_{33}\left((1-2c_1)(1-2c_3)\right) + \ldots \\ & + w_{63}\left((1-2c_{31})(1-2c_{63})\right)\end{aligned}$$

The Lower signal reaching time be defined as:

$$\begin{aligned}\Delta_{31}^l = {} & b_1 + w_0 c_0 + w_1 c_1 + w_2 c_2 + \ldots + w_{31} c_{31} \\ & + w_{32}\left((1-2c_2)\ldots(1-2c_{31})\right) \\ & + w_{33}\left((1-2c_1)(1-2c_3)\right) + \ldots \\ & + w_{63}\left((1-2c_{31})(1-2c_{63})\right)\end{aligned}$$

Subtracting above equation we get:

$$(\Delta_{31}^u - \Delta_{31}^l) = (b_1 - b_0) + (w_0 - w_0)c_0 + (w_1 - w_1)c_1 + (w_2 - w_2)c_2 + \ldots + (w_{31} - w_{31})c_{31} + (w_{32}((1-2c_2)\ldots(1-2c_{31})) - w_{32}) +$$

Above equation can be converted to another linear equation:

$$(\Delta_{31}^u) - \Delta_{31}^l = \Delta b + \Delta w_0 \cdot 0 + \Delta w_1 c_1 + \Delta w_2 c_2 + \ldots + \Delta w_{31} c_{31} + \Delta w_{32}((1-2c_2)\ldots(1-2c_{31})) + \Delta w_{33}((1-2c_1)(1-2c_3)) + \ldots$$

Now we can clearly see that if:

$$\Delta_{31}^u > 0 \quad \text{then we have} \quad r_1 = 1 \quad (1)$$

$$\Delta_{31}^u < 0 \quad \text{then we have} \quad r_1 = 0 \quad (2)$$

Therefore we can say that,

$$\frac{1 + \text{sign}(\mathbf{W}^T \cdot \phi(c) + b)}{2} = r_1$$

where,

$$\phi(c) = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{31} \\ (1-2c_0)(1-2c_1)\ldots(1-2c_{31}) \\ (1-2c_1)(1-2c_2)\ldots(1-2c_{32}) \\ \vdots \\ (1-2c_{31})(1-2c_{32}) \end{bmatrix}$$

Task 4

Dimension that we need for the linear model to predict the response remains same as required in task 2 i.e. **63**.

Task 6

The model is trained on `train.dat` and tested on `test.dat` with the different set of hyperparameters and their accuracy and training time is recorded.

3.1 Effect of changing the loss hyperparameter in LinearSVC

The **loss function** characterizes how well the model performs on a given dataset. `sklearn.svm.LinearSVC` provides two loss functions, namely: **hinge** and **squared hinge**.

We made the observations that are mentioned below using the hyperparameters `C = 1`, `tolerance = 1e-3` , `penalty = L2` and `dual = True` for **LinearSVC**.

Table 1: Loss Hyperparameters

Loss	Training Time (in seconds)	Accuracy0	Accuracy1
Hinge	22.51	0.984	0.996
Squared Hinge	24.89	0.9800	0.9978

3.2 Effect of changing the C hyperparameter in LinearSVC and Logistic Regression model

Regularization is a technique used to prevent overfitting by penalizing large coefficients in the model. The C parameter represents the inverse of regularization strength, where smaller values of C correspond to stronger regularization.
In both Logistic Regression and Linear SVC, adjusting C influences how closely the model fits the training data. Increasing C tightens the fit, potentially leading to overfitting, while decreasing C encourages simpler decision boundaries, helping generalization but risking underfitting.

We made the observations that are mentioned below :

Table 2: C Hyperparameters in LinearSVC

C value	Training Time (in seconds)	Accuracy0	Accuracy1
0.01	18.66	0.9799	0.9948
0.1	18.78	0.9803	0.9964
1	18.96	0.9800	0.998
10	18.72	0.9800	0.9987
100	18.	0.9800	0.9991

We made the observations that are mentioned below using the hyperparameters `loss = Squared Hinge`, `tolerance = 1e-3` , `penalty = L2` and `dual = False` for **LinearSVC**.

Table 3: C Hyperparameters in Logistic Regression

C value	Training Time (in seconds)	Accuracy0	Accuracy1
0.01	18.18	0.9800	0.993
0.1	18.40	0.9804	0.995
1	18.30	0.9806	0.997
10	18.40	0.9808	0.9981
100	18.85	0.9808	0.99

We made the observations that are mentioned below using the hyperparameters `solver='lbfgs'`, `max_iter=1000` ,`tol=1e-3` ,`penalty='l2'` in **LogisticRegression**.

4 Question 5

Code Submitted