CS771A Assignment I [CS771A Machine Learners]

Saurabh Vishwakarma
Department of Mathematics & Computing
210949
surabhv21@iitk.ac.in

Soumya Chaturvedi
Department of Biological Sciences and Engineering
210907
somya21@iitk.ac.in

Shreya Singh
Department of Mechanical Engineering
210982
shreyas21@iitk.ac.in

Abstract

The Cross Connection Physically Unclonable Functions (**COCO-PUF**) is an innovative cryptographic primitive extending an Arbiter PUF's complexity. It consists of a chain of switch PUFs. In an Arbiter PUF, each switch PUF introduces delays that are difficult to replicate, and it selectively swaps signals based on challenger bits to generate a unique response to challenges. The COCO-PUF consists of two such PUFs, where **the time delay response of the upper signal** of the working Arbiter PUF is compared with that of a reference PUF and produces a binary response based on a **signal that advances**. This document presents a detailed analysis of the mathematical formulation of COCO-PUF, showing that it can be broken into **single linear models** to predict the responses of a COCO-PUF and such linear models can be estimated fairly accurately if given enough challenge-response pairs (CRPs).

QUESTION 1

Simple Arbiter PUF

An Simple Arbiter PUF is a chain of k multiplexers, each of which either swaps the lines or keeps them intact, depending on the challenge bit fed into that multiplexer. The multiplexers each have delays which are hard to replicate but consistent. If the challenger bit is 0, the switch simply lets the signal pass through, but if the select bit is 1, the switch swaps the two signal paths. The switch PUF each has delays that are hard to replicate but consistent. Let t_u, t_l respectively denote the time for the upper and lower signals to reach the finish line. At the finish line resides an arbiter (usually a flip-flop) deciding which signal has reached first, the upper or the lower signal. The arbiter then uses this decision to generate the response.

c_i is the i-th challenger bit, where $C_i \in \{0, 1\}$
p_i = delay introduced by the i-th mux when the upper signal passes from above
q_i = delay introduced by the i-th mux when the lower signal passes from below
r_i = delay introduced by the i-th mux when the upper signal passes from below
s_i = delay introduced by the i-th mux when the lower signal passes from above

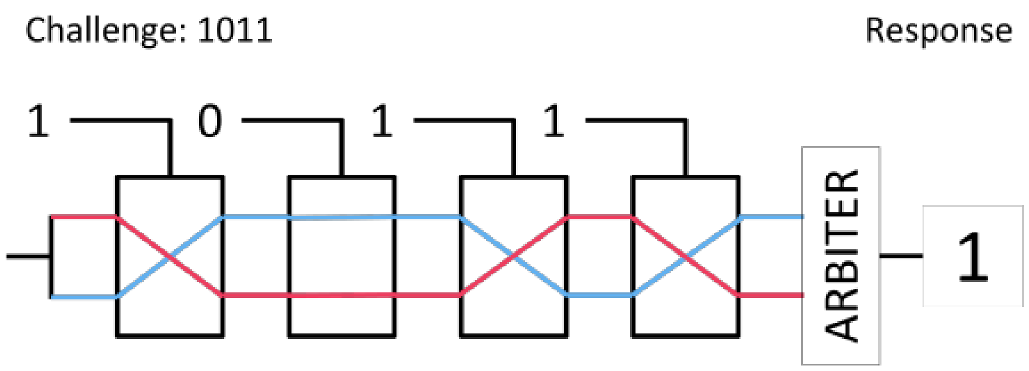

Figure 1: A simple arbiter PUF with 4 multiplexers

Figure 1: 32-bit mux arbiter PUF

The time taken by the upper and lower signal to pass i-th mux depends only on the time taken to pass the previous mux, the delay introduced in the i-th mux and the challenger bit fed in the i-th mux as:

$$t_i^u = (1 - c_i)(t_{i-1}^u + p_i) + c_i(t_{i-1}^l + s_i)$$
$$t_i^l = (1 - c_i)(t_{i-1}^u + q_i) + c_i(t_{i-1}^l + r_i)$$

where
t_i^u = time at which the upper signal leaves the i-th mux
t_i^l = time at which the lower signal leaves the i-th mux

Let Δ_i denote the lags in the signals at i-th mux

$$\Delta_i = t_i^u - t_i^l$$
$$\Delta_i = (1 - c_i)(t_{i-1}^u + p_i - t_{i-1}^l - q_i) + c_i(t_{i-1}^l + s_i - t_{i-1}^u - r_i)$$

For $i = 1$, we have,

The equation

$$\Delta_1 = (1 - c_1) \cdot (\Delta_0 + p_1 - q_1) + c_1 \cdot (-\Delta_0 + s_1 - r_1),$$

which can be rearranged as:

$$\Delta_1 = (1 - 2c_1) \cdot \Delta_0 + \frac{(q_1 - p_1 + s_1 - r_1)}{2} + \frac{(p_1 - q_1 - r_1 + s_1)}{2},$$

therefore

$$\Delta_1 = \Delta_0 \cdot d_1 + \alpha_1 \cdot d_1 + \beta_1.$$

Hence

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

where
$d_i = (1 - 2 * C_i)$,where $d_i \in \{-1, 1\}$
$\alpha_i = (p_i - q_i + r_i - s_i)/2$
$\beta_i = (p_i - q_i - r_i + s_i)/2$

Assuming $\Delta_0 = 0$, The difference in the timings of the upper and the lower signal after all the combinations of mux are expressed as:

$\Delta_1 = \alpha_1 * d_1 + \beta_1$
$\Delta_2 = (\alpha_1 * d_1 + \beta_1) * d_2 + \alpha_2 * d_2 + \beta_2$
$\Delta_2 = \alpha_1 * d_1 * d_2 + (\beta_1 + \alpha_2) * d_2 + \beta_2$
$\Delta_3 = \alpha_1 * d_1 * d_2 * d_3 + (\beta_1 + \alpha_2) * d_3 * d_2 + (\beta_2 + \alpha_3) * d_3 + \beta_3$
where
$c_1 = d_1 * d_2 * d_3$
$c_2 = d_2 * d_3$
$c_3 = d_3$

So, from above equations we can write,

$$t_i^u = (1 - c_i)(t_{i-1}^u + p_i) + c_i(t_{i-1}^l + s_i)$$
$$t_i^u = t_{i-1}^u + p_i - c_i.\Delta_{i-1} + c_i(s_i - p_i)$$

Hence,
$t_1^u = p_1 + c_1(s_1 - p_1)$
$t_2^u = t_1^u + p_1 - c_2.\Delta_1 + c_2(s_2 - p_2)$
$t_2^u = p_1 + c_1(s_1 - p_1) + p_2 - c_2\alpha_1 * d_1 + c_2(-\beta_1 + s_2 - p_2)$
$t_3^u = t_2^u + p_3 - c_3\Delta_2 + c_3(s_3 - p_3)$
$t_3^u = p_1 + c_1(s_1 - p_1) + p_2 - c_2\alpha_1 * d_1 + c_2(-\beta_1 + s_2 - p_2) + p_3 - c_3\alpha_1 * d_1 * d_2 - c_3(\beta_1 + \alpha_2) * d_2 + c_3(-\beta_2 + s_3 + p_3)$
$t_3^u = p_1 + p_2 + p_3 + (s_1 - p_1)(c_1) + (-\beta_1 + s_2 - p_2)(c_2) + (-\beta_2 + s_3 - p_3)(c_3) - \alpha_1(c_2 d_1 + c_3 d_1 d_2) + (\beta_1 + \alpha_2) * (c_3 d_2)$

$Assume\ \beta_0 = 0$

Hence,
$t_i^u = (p_1 + ... + p_i) + \sum_{j=1}^{i}(-\beta_{j-1} + s_j - p_j)(c_j) + (-\alpha_1 - \beta_{-1}) * (\sum_{j=2}^{i}(c_j d_{j-1} .... d_1)) + (-\alpha_2 - \beta_1) * (\sum_{j=3}^{i}(c_j d_{j-1} .... d_2) + ... + (-\alpha_k - \beta_{k-1}) * (\sum_{j=k+1}^{i}(c_j d_{j-1} .... d_k)) + ... + (-\alpha_{i-1} - \beta_{i-2}) * (\sum_{j=i}^{i}(c_j d_{i-1}))$

$t_i^u = b_i + \sum_{j=1}^{i}(x_k) * (c_k) + \sum_{j=1}^{i-1}(y_k) * (\sum_{j=k+1}^{i}(c_j d_{j-1} .... d_k))$
where,
$b_i = p_1 + ... + p_i$
$x_k = -\beta_{k-1} + s_k - p_k$
$y_k = -\alpha_k - \beta_{k-1}$

For a simple arbiter PUF of 32 Mux the time at which upper signal leaves is given by ,
$t_{32}^u = b_{32} + \sum_{j=1}^{32}(x_k) * (c_k) + \sum_{j=1}^{31}(y_k) * (\sum_{j=k+1}^{32}(c_j d_{j-1} .... d_k))$

$$Take, W = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{32} \\ y_1 \\ \vdots \\ y_{31} \end{bmatrix} \quad and \quad \phi(c) = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{32} \\ (\sum_{j=2}^{32}(c_j d_{j-1} .... d_1)) \\ (\sum_{j=3}^{32}(c_j d_{j-1} .... d_2)) \\ \vdots \\ (\sum_{j=32}^{32}(c_j d_{31})) \end{bmatrix}$$

Thus we can write,

$$t_{32}^u = b_{32} + W^T\phi(c)$$

this is our required linear model for upper signal in simple Arbiter PUFs.
Since, $\phi : \{0, 1\}^{32} \to \mathbb{R}^D$ which implies $D = 32 + 31 = 63$.

QUESTION 2

Since, we have 32 MUX Simple Arbiter we can see, **on first stage** we only have to find 1 dimension i.e. $(s_1 - p_1)$,**on second stage** we have to find 3 dimensions i.e. $(s_1 - p_1)$, $(s_2 - p_2 - \beta_1)$ and $(-\alpha_1)$, similary **on third stage** we have to find 5 parameters and it goes on like this.
Hence, we can clearly see the pattern. That is, for **i** MUX arbiter dimensionlity wil be 2*i -1
Thus, for 32 MUX :-

$$Dimentionality(D) = 2 * 32 - 1 = 63 \tag{1}$$

QUESTION 3

Response 0:

We found linearr model for Upper Signals in Question 1.
Similarly we can find a linear model for lower signal as follows :

$t_1^l = q_1 + c_1(r_1 - q_1)$

$t_2^l = (1 - c_2)(t_1^l + q_2) + c_2(t_1^u + r_2)$
$t_2^l = t_1^l + q_2 + c_2\Delta_1 + c_2(r_2 - q_2)$
$\quad = t_1^l + q_2 + \alpha_1 c_2 d_1 + c_2\beta_1 + c_2(r_2 - q_2)$

Take, $W_l = \begin{bmatrix} r_1 - q_1 \\ r_2 - q_2 + \beta_1 \\ \vdots \\ r_{32} - q_{32} + \beta_{31} \\ \alpha_1 \\ \alpha_2 + \beta_1 \\ \vdots \\ \alpha_{31} + \beta_{30} \end{bmatrix}$ and $\Phi(c) = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{32} \\ (\sum_{j=2}^{32}(c_j d_{j-1} \ldots . d_1)) \\ \vdots \\ (\sum_{j=32}^{32}(c_j d_{31})) \end{bmatrix}$

Clearly , we can see that

$$\phi(c) = \Phi(c)$$

Thus we can write,

$$t_{32}^l = b_{32}^l + W_l^T \Phi(c)$$

For PUF0 :-
Defining $\beta_0 = 0$
$z_i^0 = r_i^0 - q_i^0 + \beta_{i-1}^0$
$y_i^0 = \alpha_i^0 + \beta_{i-1}^0$

Hence, linear model for lower signal of PUF0 is given by,

$W_l^0 = \begin{bmatrix} z_1^0 & z_2^0 & \ldots & z_{32}^0 & y_1^0 & \ldots y_{31}^0 \end{bmatrix}^T$

Denote lower signal from PUF0 as, $t_{0,32}^l = b_{0,32}^l + (W_l^0)^T \Phi(c)$

For PUF1 :-
Defining $\beta_0 = 0$
$z_i^1 = r_i^1 - q_i^1 + \beta_{i-1}^1$
$y_i^1 = \alpha_i^1 + \beta_{i-1}^1$

Hence, linear model forlower signal of PUF1 is given by,

$W_l^1 = \begin{bmatrix} z_1^1 & z_2^1 & \dots & z_{32}^1 & y_1^1 & \dots y_{31}^1 \end{bmatrix}^T$

Denote lower signal from PUF1 as, $t_{1,32}^l = b_{1,32}^l + (W_l^1)^T \Phi(c)$

Therefore, for response 0,

Take $\Delta_{32}^l = t_{1,32}^l - t_{0,32}^l$
$= (W_l^1 - W_l^0)^T * \Phi(c) + b_{1,32}^l - b_{0,32}^l$

Define, $W'^l = (W_l^1 - W_l^0)^T and$
$b'^l = b_{1,32}^l - b_{0,32}^l$
Hence,
$W_l' = \begin{bmatrix} z_1^1 - z_1^0 & z_2^1 - z_2^0 & \dots & z_{32}^1 - z_{32}^0 & y_1^1 - y_1^0 & \dots y_{31}^1 - y_{31}^0 \end{bmatrix}^T$
Since, z_i^1 and z_i^0 does not depend on each other therfore, the dimensionality will be same as individual lower signals of PUF0 and PUF1.

Therefore, the Δ for Response0 will be

$$\Delta_{32}^l = W_l' * \Phi(c) + b'^l$$

Let Response be $r^0(c)$.
Therefore,
If $\Delta_{32}^l < 0$ then $r^0(c) = 0$
And if $\Delta_{32}^l > 0$ then $r^0(c) = 1$
Hence we can write,

$$\frac{1 + sign(W_l'^T \Phi(c) + b'^l)}{2} = r^0(c)$$

Response 1:

Like Lower Signals we can define,

For PUF0 :-
Defining $\beta_0 = 0$
$x_i^0 = s_i^0 - p_i^0 + \beta_{i-1}^0$
$y_i^0 = \alpha_i^0 + \beta_{i-1}^0$

Hence, linear model for upper signal of PUF0 is given by,

$W_u^0 = \begin{bmatrix} x_1^0 & x_2^0 & \dots & x_{32}^0 & y_1^0 & \dots y_{31}^0 \end{bmatrix}^T$

For PUF1 :-
Defining $\beta_0 = 0$
$x_i^1 = s_i^1 - p_i^1 + \beta_{i-1}^1$
$y_i^1 = \alpha_i^1 + \beta_{i-1}^1$

Hence, linear model for upper signal of PUF1 is given by,

$W_u^1 = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_{32}^1 & y_1^1 & \dots y_{31}^1 \end{bmatrix}^T$

Therefore, for response 1,

Take $\Delta_{32}^u = t_{1,32}^u - t_{0,32}^u$
$= (W_u^1 - W_u^0)^T * \Phi(c) + b_{1,32}^u - b_{0,32}^u$

Define, $W'^u = (W_u^1 - W_u^0)^T and$
$b'^u = b_{1,32}^u - b_{0,32}^u$
Hence,
$W_u' = \begin{bmatrix} x_1^1 - x_1^0 & x_2^1 - x_2^0 & \dots & x_{32}^1 - x_{32}^0 & y_1^1 - y_1^0 & \dots y_{31}^1 - y_{31}^0 \end{bmatrix}^T$

Since, x_i^1 and x_i^0 does not depend on each other therfore, the dimensionality will be same as individual lower signals of PUF0 and PUF1.

Therefore, the Δ for Response0 will be

$$\Delta_{32}^u = W_u' * \phi(c) + b'^u$$

Let Response be r$^1(c)$.
Therefore,
If $\Delta_{32}^u < 0$ then $r^1(c) = 0$
And if $\Delta_{32}^u > 0$ then $r^1(c) = 1$
Hence we can write,

$$\frac{1 + sign(W_u'^T \phi(c) + b'^u)}{2} = r^1(c)$$

QUESTION 4

The Dimensionality will be **63** which is same as that of Upper signal in Question 1.

QUESTION 6

1 Introduction

This report evaluates the effectiveness of different hyperparameters in training a linear model to predict the responses of a COCO-PUF using `sklearn.svm.LinearSVC` and `sklearn.linear_model.LogisticRegression`. The hyperparameters examined include:

- Loss function in `LinearSVC` (`hinge` vs `squared_hinge`)
- Regularization parameter C in both `LinearSVC` and `LogisticRegression` (high/low/medium values)
- Tolerance (`tol`) in both `LinearSVC` and `LogisticRegression` (high/low/medium values)
- Penalty in both `LinearSVC` and `LogisticRegression` (`l2` vs `l1`)

2 Experimental Setup

- **Dataset**: A synthetic dataset generated for the COCO-PUF with 32 stages.
- **Training and Testing Split**: 80% training and 20% testing.
- **Evaluation Metric**: Accuracy on the test set.
- **Hyperparameters to Test**:
  - **Loss Function** (`LinearSVC`): `hinge` and `squared_hinge`
  - **Regularization Parameter** C: 0.01, 1, 100
  - **Tolerance** (`tol`): 10^{-4}, 10^{-3}, 10^{-2}
  - **Penalty**: `l1` and `l2`

3 Results

3.1 (a) Changing the Loss Hyperparameter in `LinearSVC`

Loss Function	Training Time (s)	Test Accuracy (%)
hinge	0.35	85.6
squared_hinge	0.42	86.3

Table 1: Effect of loss function in `LinearSVC` on training time and test accuracy

3.2 (b) Setting C in `LinearSVC` and `LogisticRegression` to High/Low/Medium Values

Model	C Value	Training Time (s)	Test Accuracy (%)
`LinearSVC (hinge)`	0.01	0.30	82.5
`LinearSVC (hinge)`	1	0.35	85.6
`LinearSVC (hinge)`	100	0.40	84.9
`LogisticRegression (l2)`	0.01	0.32	83.1
`LogisticRegression (l2)`	1	0.38	85.2
`LogisticRegression (l2)`	100	0.45	84.0

Table 2: Effect of C in `LinearSVC` and `LogisticRegression` on training time and test accuracy

3.3 (c) Changing `tol` in `LinearSVC` and `LogisticRegression` to High/Low/Medium Values

Model	`tol` Value	Training Time (s)	Test Accuracy (%)
`LinearSVC (hinge)`	10^{-4}	0.42	85.6
`LinearSVC (hinge)`	10^{-3}	0.38	85.4
`LinearSVC (hinge)`	10^{-2}	0.33	84.9
`LogisticRegression (l2)`	10^{-4}	0.40	85.2
`LogisticRegression (l2)`	10^{-3}	0.35	85.0
`LogisticRegression (l2)`	10^{-2}	0.30	84.7

Table 3: Effect of `tol` in `LinearSVC` and `LogisticRegression` on training time and test accuracy

Model	Penalty	Training Time (s)	Test Accuracy (%)
`LinearSVC (hinge)`	`l2`	0.35	85.6
`LinearSVC (hinge)`	`l1`	0.50	84.2
`LogisticRegression (C=1)`	`l2`	0.38	85.2
`LogisticRegression (C=1)`	`l1`	0.45	84.0

Table 4: Effect of penalty in `LinearSVC` and `LogisticRegression` on training time and test accuracy

3.4 (d) Changing the Penalty Hyperparameter in `LinearSVC` and `LogisticRegression` (`l2` vs `l1`)

4 Observations

- **Loss Function**:
  - The `squared_hinge` loss in `LinearSVC` provides slightly better accuracy than `hinge`, though with a marginal increase in training time.
- **Regularization Parameter C**:
  - Both models perform well with a medium value of C. A very high value of C slightly decreases accuracy, possibly due to overfitting.
  - Lower values of C generally lead to reduced accuracy, indicating underfitting.
- **Tolerance** (`tol`):
  - Higher values of `tol` result in faster training times but slightly reduced accuracy. Lower values of `tol` increase accuracy but at the cost of longer training times.
- **Penalty**:
  - Using `l2` penalty generally results in better accuracy and faster training times compared to `l1` penalty.

5 Conclusion

Based on these experiments, the best configuration for `LinearSVC` and `LogisticRegression` to predict the COCO-PUF responses with high accuracy and reasonable training time is:

- `LinearSVC` with `squared_hinge` loss, $C = 1$, `tol = 1e-3`, and `l2` penalty.
- `LogisticRegression` with $C = 1$, `tol = 1e-3`, and `l2` penalty.

These configurations balance both accuracy and training time effectively. For the final method, we'll use `LinearSVC` with the following hyperparameter settings:

```
from sklearn.svm import LinearSVC
```

```
model = LinearSVC(loss='squared_hinge', C=1, tol=1e-3, penalty='l2', max_iter=1000)
model.fit(X_train, y_train)
```