CS771 : Introduction to Machine Learning
Assignment - 1

Aadish Patidar
220005
aadishp22@iitk.ac.in

Agam Pipersenia
220086
agamp22@iitk.ac.in

Ashwin Chaubey
220245
ashwinc22@iitk.ac.in

Dhruv Mittal
220363
dhruvm22@iitk.ac.in

Suman Kumar Maharana
232090906
sumanm20@iitk.ac.in

Yash Chauhan
221217
yashc22@iitk.ac.in

Abstract

This report investigates the security and predictability of a newly proposed Physical Unclonable Function (PUF) variant called the Cross-Connection PUF (COCO-PUF). Traditional arbiter PUFs, known for their vulnerability to linear modeling attacks, serve as the foundation for the COCO-PUF. Unlike arbiter PUFs, the COCO-PUF employs two arbiter PUFs and cross-connects their output signals, introducing a perceived increase in complexity and security. This study aims to determine whether COCO-PUF's cross-connection structure significantly enhances its resistance to linear modeling attacks. We derive mathematical formulations demonstrating that, similar to arbiter PUFs, linear models can accurately predict COCO-PUF responses.

Using a dataset of 40,000 challenge-response pairs (CRPs), we implement linear classifiers, including Logistic Regression and Linear Support Vector Classification (LinearSVC), to predict the COCO-PUF responses. Our experimental results indicate that both models achieve high accuracy, thus challenging the security assumptions of COCO-PUF. The findings highlight that despite the added complexity, COCO-PUF remains susceptible to linear modeling attacks, and therefore, similar cryptographic measures as those used for traditional arbiter PUFs should be considered.

1 Derivation for a Simple Arbiter PUF

The time it takes for the signal to propagate through the i-th multiplexer (MUX) pair in an arbiter PUF can be expressed as follows. Let $t^u_{i,1}$ and $t^l_{i,1}$ represent the times at which the signal departs from the i-th MUX pair for the upper and lower lines, respectively. Expressing t^u_i and t^l_i in terms of t^u_{i-1}, t^l_{i-1}, and the challenge bit c_i, we obtain:

$$t^u_{2,0} = (1 - c_2) \cdot (t^u_{1,0} + p_{2,0}) + c_2 \cdot (t^l_1 + s_2)$$

$$t^l_{2,0} = (1 - c_2) \cdot (t^l_{1,0} + q_{2,0}) + c_2 \cdot (t^u_1 + r_2)$$

For simplicity, we can omit the subscript 0, leading to:

$$t^u_2 = (1 - c_2) \cdot (t^u_1 + p_2) + c_2 \cdot (t^l_1 + s_2)$$

$$t^l_2 = (1 - c_2) \cdot (t^l_1 + q_2) + c_2 \cdot (t^u_1 + r_2)$$

To simplify the analysis, we define two new variables, A_2 and B_2, as follows:

$$A_2 = t^u_2 + t^l_2$$

$$B_2 = t^u_2 - t^l_2$$

From these definitions, we can express t^u_2 as:

$$t^u_2 = \frac{A_2 + B_2}{2}$$

Next, we derive expressions for A_2 and B_2:

$$A_2 = (1 - c_2) \cdot (A_1 + p_2 + q_2) + c_2 \cdot (A_1 + s_2 + r_2) + A_1$$

Simplifying the above expression, we get:

$$A_2 = (1 - c_2) \cdot (p_2 + q_2) + c_2 \cdot (s_2 + r_2) + A_1$$

Similarly, for B_2, we have:

$$B_2 = (1 - c_2) \cdot (B_1 + p_2 - q_2) + c_2 \cdot (-B_1 + s_2 - r_2)$$

To further simplify, we define the following parameters:

$$d_i = (1 - 2 \cdot c_i)$$

$$\alpha_i = \frac{p_i - q_i + r_i - s_i}{2}$$

$$\beta_i = \frac{p_i - q_i - r_i + s_i}{2}$$

$$g_i = \frac{p_i + q_i + r_i - s_i}{2}$$

$$f_i = \frac{s_i + r_i - (p_i + q_i)}{2}$$

Using these definitions, we rewrite A_i and B_i as:

$$A_i = (1 - c_i) \cdot (p_i + q_i) + c_i \cdot (s_i + r_i) + A_{i-1} \quad where\ A_0 = 0$$

$$B_i = (1 - 2 \cdot c_i) \cdot (B_{i-1}) + c_i \cdot (s_i - r_i - (p_i - q_i)) + p_i - q_i$$

Simplifying further, we get:

$$B_i = d_i \cdot B_{i-1} + d_i \cdot \alpha_i + \beta_i$$

Thus, the equations for A_i and B_i can be expressed as:

$$A_i = p_i + q_i + c_i \cdot (-(p_i + q_i) + (s_i + r_i)) + A_{i-1}$$

$$A_i = A_{i-1} - d_i \cdot f_i + g_i$$

$$B_i = d_i \cdot B_{i-1} + d_i \cdot \alpha_i + \beta_i$$

By induction, for $i = 32$ (considering the PUF has 32 stages), we obtain:

$$A_{32} = w'_1 \cdot x'_1 + w'_2 \cdot x'_2 + \cdots + w'_{32} \cdot x'_{32} + b'$$

$$B_{32} = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_{32} \cdot x_{32} + b$$

To illustrate, let's derive A_1 explicitly:

$$A_1 = (1 - c_1) \cdot p_i + c_1 \cdot s_1 + (1 - c_1) \cdot q_1 + c_1 \cdot r_1$$

$$A_1 = (1 - c_1) \cdot (p_1 + q_1) + c_1 \cdot (r_1 + s_1)$$

$$A_1 = p_1 + q_1 + c_1 \cdot (r_1 + s_1 - (p_1 + q_1))$$

$$A_1 = g_1 - f_1 + f_1 \cdot (1 - d_i)$$

$$A_1 = g_1 - f_1 \cdot d_1$$

Defining the vectors x_i and x'_i as:

$$x_i = d_i d_{i+1} \cdots d_{32}, \quad x'_i = -d_i$$

and the weights w_i and w'_i as:

$$w_i = \alpha_i, \quad w'_i = f_i$$

we have:

$$w_i = \alpha_i + \beta_{i-1} \quad (where i = 1, 2, \dots)$$

Finally, we define the bias terms b and b' as:

$$b = \beta_{32}, \quad b' = \sum_{i=1}^{32} g_i$$

Therefore, the time it takes for the upper signal to reach the finish line for the 32-bit arbiter PUF can be expressed as:

$$t_{32}^u = \frac{A_{32} + B_{32}}{2}$$

$$t_{32}^u = \sum_{i=1}^{32} \frac{w_i \cdot x_i + w_i' \cdot x_i' + b + b'}{2}$$

$$t_{32}^u = \frac{w^T \cdot x + b + w'^T \cdot x' + b'}{2}$$

$$t_{32}^u = \frac{(w'^T \cdot x' + b') - (w^T \cdot x + b)}{2}$$

This establishes that there exists a linear model that can predict the time t_{32}^u for the upper signal in an arbiter PUF. The feature map Φ transforms the 32-bit challenge into a suitable higher-dimensional space:

$\Phi : \{0,1\}^{32} \to \mathbb{R}^D$

Given a challenge C, the feature map can be represented as $\{x, x'\}$ where x and x' are derived from the challenge bits and the associated PUF-specific constants.

2 Dimensionality for a Simple Arbiter PUF

The time it takes for the upper signal to reach the finish line in a 32-stage arbiter PUF can be expressed as:

$$t_{32}^u = \frac{w^T \cdot x + b + w'^T \cdot x' + b'}{2}$$

This equation can be expanded as follows:

$$t_{32}^u = \frac{w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_{32} \cdot x_{32} + b + w_1' \cdot x_1' + w_2' \cdot x_2' + \cdots + w_{32}' \cdot x_{32}' + b'}{2}$$

Where w and w' are weight vectors, x and x' are feature vectors, and b and b' are bias terms.

Given the property of the feature vectors, we assume x_i can be related to x_i' as follows:

$$x_i = (-1)^i \cdot x_i' \quad so \quad x_{32} = -x_{32}'$$

Substituting $x_{32} = -x_{32}'$ into the expression for t_{32}^u, we get:

$$t_{32}^u = \frac{w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_{31} \cdot x_{31} + w_{32} \cdot (-x_{32}') + b + w_1' \cdot x_1' + w_2' \cdot x_2' + \cdots + w_{32}' \cdot x_{32}' + b'}{2}$$

Simplifying this, we have:

$$t_{32}^u = \frac{w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_{31} \cdot x_{31} + b + w_1' \cdot x_1' + w_2' \cdot x_2' + \cdots + (w_{32}' - w_{32}) \cdot x_{32}' + b'}{2}$$

The above expression indicates that the final term for x_{32}' is modified by subtracting w_{32} from w_{32}'. Hence, the overall feature vector in the transformed space and its corresponding weight vector must be considered.

The dimensionality of the model can be deduced from the number of unique weight and feature terms present in the final expression. The weight vectors w and w' each have 32 components, while the terms involving x_i and x'_i also appear distinctly, giving a total of:

- 31 terms involving $w_i \cdot x_i$ - 31 terms involving $w'_i \cdot x'_i$ - 1 term involving $(w'_{32} - w_{32}) \cdot x'_{32}$ - 2 bias terms b and b'

Therefore, the dimensionality of the model that predicts the arrival time of the upper signal in a 32-stage arbiter PUF is:

$$31 + 31 + 1 = 63$$

Thus, we conclude:

$$Dimensionality = 63$$

3 Derivation for a COCO-PUF

In a COCO-PUF, we analyze the responses based on the comparison of the upper and lower signal propagation times across the arbiter stages. Specifically, we define the responses R_0 and R_1 as follows:

First, the response R_0 is determined by comparing the time differences of the lower signals between two stages:

$$R_0 = bool(t^l_{32,0} - t^l_{32,1} > 0)$$

Similarly, R_1 is defined by comparing the time differences of the upper signals:

$$R_1 = bool(t^u_{32,0} - t^u_{32,1} > 0)$$

Next, we express R_0 using the weights and feature vectors derived from the signal propagation times in the arbiter PUF. Given that t^l_{32} and t^u_{32} are linear functions of the feature vectors and weights, we have:

$$R_0 = bool(w_0^T x'_0 + b'_0 - w_0^T x_0 + b_0 - w_1'^T x'_1 + b'_1 + w_1^T x_1 + b_1 > 0)$$

To represent R_0 in a more analytical form, we use the signum function $sgn(x)$, which is defined as $sgn(x) = \frac{x}{|x|}$. Thus, we can represent R_0 as:

$$R_0 = \frac{1 + sgn(w_0^T x'_0 + b'_0 - w_0^T x_0 + b_0 - w_1'^T x'_1 + b'_1 + w_1^T x_1 + b_1)}{2}$$

Here, $\Phi : \{0,1\}^{32} \to \mathbb{R}^D$ denotes the feature map that transforms the 32-bit challenge into a higher-dimensional space. The challenge C is mapped to the feature vectors $\{-x_0, x'_0, x_1, -x'_1\}$.

Similarly, for the response R_1, we have:

$$R_1 = \frac{1 + sgn(w_0^T x'_0 + b'_0 - w_0^T x_0 + b_0 - w_1'^T x'_1 + b'_1 + w_1^T x_1 + b_1)}{2}$$

Again, $\Phi : \{0,1\}^{32} \to \mathbb{R}^D$ represents the feature map, and the challenge C is mapped to the feature vectors $\{x_0, x'_0, -x_1, -x'_1\}$.

In summary, the COCO-PUF responses R_0 and R_1 are determined by comparing the weighted sums of the feature vectors corresponding to different signal propagation paths. The use of the signum function allows for a clear, non-linear decision boundary, which is useful for modeling and analyzing the PUF behavior.

Response 0 and 1 both:
The term x_{32} will be the same in all four quantities, hence the total dimensionality is $4 \times 31 + 1 = 125$.

4 Dimensionality for a COCO-PUF

The dimensionality of the COCO-PUF can be derived from the expressions for R_0 and R_1. Each response is influenced by the challenge bits transformed into higher-dimensional feature vectors. To understand the dimensionality, we consider the components involved in the responses R_0 and R_1.

First, let's restate the expressions for R_0 and R_1:

$$R_0 = \frac{1 + sgn(w_0^T x_0' + b_0' - w_0^T x_0 + b_0 - w_1'^T x_1' + b_1' + w_1^T x_1 + b_1)}{2}$$

$$R_1 = \frac{1 + sgn(w_0^T x_0' + b_0' - w_0^T x_0 + b_0 - w_1'^T x_1' + b_1' + w_1^T x_1 + b_1)}{2}$$

Here, the feature map $\Phi : \{0,1\}^{32} \to \mathbb{R}^D$ transforms the 32-bit challenge into a set of higher-dimensional feature vectors. The challenges C are mapped to the feature vectors $\{x_0, x_0', x_1, -x_1'\}$ for R_0 and $\{x_0, x_0', -x_1, -x_1'\}$ for R_1.

To derive the dimensionality, we note that each term $w_i^T x_i$ involves the dot product of a weight vector and a feature vector. Considering both R_0 and R_1, we need to account for the number of unique terms involved in these expressions.

For each R_i (where $i = 0, 1$), there are: - 32 weight vectors w_0 and w_1 associated with x_0 and x_1 - 32 weight vectors w_0' and w_1' associated with x_0' and x_1' - 32 biases b_0, b_0', b_1, and b_1'

Each weight vector and bias can be considered a separate dimension. Therefore, we calculate the dimensionality as follows: rs

Adding these up, we get:

$$Dimensionality = 32 + 32 + 32 + 32 - 1 - 1 - 1 = 125$$

However, in practice, some of these dimensions might overlap or be redundant, leading to a slightly reduced effective dimensionality. In this case, after accounting for redundancies, the effective dimensionality of the COCO-PUF is 125.

Therefore, the dimensionality of the model that predicts the responses of the COCO-PUF is 125.

5 Solution for Part 5

*Zipped **Solution** to Assignment 1*

6 Performance Analysis for Models

(a) Performance Comparison of LinearSVC with Different Loss Functions:

Loss function	Total Features	Model Train Time (s)	Test accuracy 0	Test Accuracy 1
Hinge	32	6.5096	0.9821	0.9937
Squared Hinge	32	9.5952	0.9857	0.9944

Analysis:

From the table, we observe the following:

- **Training Time:** The model with the Hinge loss function has a shorter training time of 6.5096 seconds compared to the Squared Hinge loss function, which has a training time of 9.5952 seconds. This indicates that the Hinge loss function is computationally less expensive.

- **Test Accuracy for Class 0:** The test accuracy for class 0 is slightly lower for the Hinge loss function (0.9821) compared to the Squared Hinge loss function (0.9857). This suggests that the Squared Hinge loss function provides a better fit for class 0.
- **Test Accuracy for Class 1:** The test accuracy for class 1 is marginally higher for the Squared Hinge loss function (0.9944) compared to the Hinge loss function (0.9937). This indicates that the Squared Hinge loss function also performs slightly better for class 1.

In conclusion, while the Squared Hinge loss function results in slightly higher test accuracy for both classes, it comes at the cost of increased training time. Depending on the specific requirements and constraints of the application, one might prefer the Hinge loss function for faster training or the Squared Hinge loss function for slightly better accuracy.

(b) Performance Comparison based on the value of C

Analysis for LinearSVC with different C values:

To assess the impact of the regularization parameter C on the performance of 'LinearSVC', we evaluated the model with low, medium, and high C values.

Table 1: Performance Comparison of LinearSVC with Different C Values

C Value	Total Features	Model Train Time (s)	Test Accuracy 0	Test Accuracy 1
Low	32	11.4641	0.9817	0.9929
Medium	32	10.2671	0.9860	0.9944
High	32	10.0647	0.9841	0.9938

- **Training Time:** As C increases, the training time decreases slightly from 11.4641 seconds for low C to 10.0647 seconds for high C. This suggests that higher regularization (lower C) leads to more computational complexity.
- **Test Accuracy for Class 0:** The test accuracy for class 0 is highest for the medium C value (0.9860), followed by the high C value (0.9841), and lowest for the low C value (0.9817). This indicates that the model's performance improves with increased regularization up to a certain point.
- **Test Accuracy for Class 1:** Similarly, the test accuracy for class 1 is highest for the medium C value (0.9944), followed by the high C value (0.9938), and lowest for the low C value (0.9929). This trend suggests that a medium level of regularization yields the best overall performance.

Analysis for LogisticRegression with different C values:

We also evaluated the performance of 'LogisticRegression' with varying C values.

Table 2: Performance Comparison of LogisticRegression with Different C Values

C Value	Total Features	Model Train Time (s)	Test Accuracy 0	Test Accuracy 1
Low	32	13.6300	0.9777	0.9896
Medium	32	13.1820	0.9806	0.9926
High	32	13.1017	0.9824	0.9935

- **Training Time:** The training time decreases slightly as C increases, from 13.6300 seconds for low C to 13.1017 seconds for high C. This indicates that higher values of C (lower regularization) require less training time.
- **Test Accuracy for Class 0:** The test accuracy for class 0 increases from 0.9777 for low C to 0.9824 for high C. This demonstrates that higher values of C (lower regularization) improve the model's accuracy for class 0.

- **Test Accuracy for Class 1:** Similarly, the test accuracy for class 1 increases from 0.9896 for low C to 0.9935 for high C. This trend suggests that lower regularization leads to better performance for class 1 as well.

In conclusion, the performance of both 'LinearSVC' and 'LogisticRegression' is influenced by the value of the regularization parameter C. For 'LinearSVC', a medium C value provides the best balance between training time and accuracy. For 'LogisticRegression', higher C values (lower regularization) result in better accuracy for both classes, though the training time remains relatively stable.

(c) Effect of Variation in Tolerance for LinearSVC and LogisticRegression Models

For part (c) of the analysis, we evaluate the performance of the LinearSVC and LogisticRegression models with varying values of the `tol` hyperparameter. The `tol` hyperparameter controls the tolerance for the optimization algorithm, with lower values potentially leading to longer training times but potentially improved convergence and accuracy.

Table 3: Performance Comparison of LinearSVC and LogisticRegression with Different Tolerance

Model	Tolerance	Total Features	Model Train Time (s)	Test Accuracy 0	Test Accuracy 1
LinearSVC	Low	32	8.9728	0.9861	0.9942
LinearSVC	Medium	32	9.6238	0.9854	0.9945
LinearSVC	High	32	9.8234	0.9862	0.9945
LogisticRegression	Low	32	13.2498	0.9806	0.9926
LogisticRegression	Medium	32	14.2537	0.9806	0.9926
LogisticRegression	High	32	13.1829	0.9806	0.9926

LinearSVC:

- **Training Time:** The training time slightly increases with higher tolerance values, ranging from 8.9728 seconds for low tolerance to 9.8234 seconds for high tolerance. This suggests that lower tolerance values, which require more precise convergence, result in faster training times.
- **Test Accuracy for Class 0:** The test accuracy for class 0 remains relatively stable across different tolerance values, with a slight variation. The accuracy ranges from 0.9854 for medium tolerance to 0.9862 for high tolerance, indicating that tolerance has a minimal effect on the performance for class 0.
- **Test Accuracy for Class 1:** The test accuracy for class 1 is also consistent across different tolerance values, ranging from 0.9942 to 0.9945. This stability suggests that the tolerance parameter does not significantly impact the model's ability to correctly classify class 1 instances.

LogisticRegression:

- **Training Time:** The training time varies with different tolerance values, from 13.1829 seconds for high tolerance to 14.2537 seconds for medium tolerance. Unlike LinearSVC, LogisticRegression does not show a clear trend in training time relative to tolerance.
- **Test Accuracy for Class 0:** The test accuracy for class 0 remains the same across all tolerance values, at 0.9806. This consistency indicates that tolerance does not affect the performance for class 0 in LogisticRegression.
- **Test Accuracy for Class 1:** Similarly, the test accuracy for class 1 is constant at 0.9926 for all tolerance values, showing that the tolerance parameter does not influence the model's performance for class 1.

In conclusion, the tolerance parameter has a minor impact on the training time and accuracy of the LinearSVC model, with lower tolerance values resulting in slightly faster training times. However, the tolerance parameter does not significantly affect the performance of the LogisticRegression model, as both the training time and accuracy remain stable across different tolerance values.

(d) Effect of Variation in Penalty (Regularization) Hyperparameter for LinearSVC and LogisticRegression Models

In part (d) of the analysis, we evaluate the performance of the LinearSVC and LogisticRegression models with different penalty (regularization) hyperparameters. The penalty hyperparameter controls the type of regularization applied to the model to prevent overfitting. We compare the performance of the models using L1 and L2 regularization.

Table 4: Performance Comparison of LinearSVC and LogisticRegression with Different Penalty

Model	Penalty	Total Features	Model Train Time (s)	Test Accuracy 0	Test Accuracy 1
LinearSVC	L1	32	121.6582	0.9861	0.9943
LinearSVC	L2	32	12.0927	0.9859	0.9945
LogisticRegression	L1	32	78.0598	0.9848	0.9939
LogisticRegression	L2	32	13.1829	0.9806	0.9926

LinearSVC:

- **Training Time:** The training time for LinearSVC with L1 penalty is significantly higher at 121.6582 seconds compared to 12.0927 seconds for L2 penalty. This indicates that the L1 penalty, which promotes sparsity in the model coefficients, requires more computational effort and time to converge.
- **Test Accuracy for Class 0:** The test accuracy for class 0 is 0.9861 with L1 penalty and 0.9859 with L2 penalty. This slight difference suggests that both penalties provide comparable performance for class 0.
- **Test Accuracy for Class 1:** The test accuracy for class 1 is 0.9943 with L1 penalty and 0.9945 with L2 penalty. Similar to class 0, the performance is nearly identical, indicating that the choice of penalty does not significantly impact the accuracy for class 1.

LogisticRegression:

- **Training Time:** The training time for LogisticRegression with L1 penalty is 78.0598 seconds, while it is 13.1829 seconds with L2 penalty. This shows that L1 regularization is more computationally intensive than L2 regularization for LogisticRegression as well.
- **Test Accuracy for Class 0:** The test accuracy for class 0 is 0.9848 with L1 penalty and 0.9806 with L2 penalty. The higher accuracy with L1 penalty suggests that L1 regularization might be more effective in preventing overfitting for class 0.
- **Test Accuracy for Class 1:** The test accuracy for class 1 is 0.9939 with L1 penalty and 0.9926 with L2 penalty. Similar to class 0, the L1 penalty provides slightly better performance for class 1.