CS771 : Introduction to Machine Learning
Mini Project - 1

Name	Roll No	Email
Prashant Verma	220806	prashantv22@iitk.ac.in
Nitin Gautam	220733	niting22@iitk.ac.in
Hariom Singh	220422	harioms22@iitk.ac.in
Prakhar Nanda	220784	prakharn22@iitk.ac.in
Gaurav Kataria	210387	gkataria21@iitk.ac.in
Isha Verma	210454	ishaverma21@iitk.ac.in

Abstract

This report examines the security of Physical Unclonable Functions (PUFs) through two key investigations. First, we analyze the Multi-level PUF (ML-PUF) and demonstrate that, despite its complex design—which combines two interconnected arbiter PUFs and an XOR-based response mechanism—its behavior can still be predicted using linear models. We developed a method to transform the 8-bit challenges into a structured format suitable for linear classification and determined the minimum feature dimensions necessary for accurate prediction. Our analysis is backed by experimental validation using a dataset of 6,400 challenge-response pairs for training and 1,600 for testing, which showcases the effectiveness of our approach in compromising the security of the ML-PUF.
Second, we address the challenge of recovering the internal delay parameters of an arbiter PUF when only its linear model is known. By framing this as a constrained optimization problem, we derived a solution that reconstructs valid, non-negative delay values consistent with the given model. Our results confirm that even when the internal delays are hidden, they can be accurately inferred, further highlighting vulnerabilities in PUF designs. Together, these findings emphasize the need for more robust PUF architectures that are resistant to machine learning and mathematical analysis.earning and mathematical analysis.

1)Feature Map and Linear Model for ML-PUF

The response of an ML-PUF is modeled as a linear function over a feature map derived from the input challenge vector **c** $\in \{0,1\}^8$. Consider a k-stage Arbiter PUF with challenges $c_i \in \{0,1\}$ and multiplexer delays p_i, q_i, r_i, s_i for each stage i.

Signal Propagation

Let t_i^u and t_i^l represent the arrival times of upper and lower signals at stage i:

$$t_0^u = \text{initial upper delay}$$
$$t_0^l = \text{initial lower delay}$$

For each subsequent stage $i \geq 1$:

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

Time Difference

Define $\Delta_i = t_i^u - t_i^l$ as the time difference at stage i:

$$\begin{aligned} \Delta_i &= (1-c_i)(\Delta_{i-1} + p_i - q_i) + c_i(-\Delta_{i-1} + s_i - r_i) \\ &= (1-2c_i)\Delta_{i-1} + (1-c_i)(p_i - q_i) + c_i(s_i - r_i) \end{aligned}$$

Let $d_i = 1 - 2c_i \in \{-1, +1\}$ and define:

$$\begin{aligned} \alpha_i &= \frac{p_i - q_i + r_i - s_i}{2} \\ \beta_i &= \frac{p_i - q_i - r_i + s_i}{2} \end{aligned}$$

Then the recurrence simplifies to:

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

Recursive Solution

Solving recursively with $\Delta_{-1} = 0$:

$$\begin{aligned} \Delta_0 &= \alpha_0 d_0 + \beta_0 \\ \Delta_1 &= d_1 \Delta_0 + \alpha_1 d_1 + \beta_1 \\ &= \alpha_0 d_1 d_0 + (\alpha_1 + \beta_0) d_1 + \beta_1 \\ &\vdots \\ \Delta_k &= \sum_{i=0}^{k-1} \left(\alpha_i \prod_{j=i}^{k-1} d_j \right) + \sum_{i=0}^{k-1} \left(\beta_i \prod_{j=i+1}^{k-1} d_j \right) + \beta_{k-1} \end{aligned}$$

Linear Model Formulation

Define the transformed features:

$$x_i = \prod_{j=i}^{k-1} d_j \quad \text{for } i = 0, \ldots, k-1$$

Then the final time difference becomes:

$$\Delta_k = \mathbf{w}^T \mathbf{x} + b$$

where the weights and bias are:

$$\begin{aligned} w_0 &= \alpha_0 \\ w_i &= \alpha_i + \beta_{i-1} \quad \text{for } i = 1, \ldots, k-1 \\ b &= \beta_{k-1} \end{aligned}$$

The PUF response is:

$$r = \frac{1 + \text{sign}(\mathbf{w}^T \mathbf{x} + b)}{2}$$

Linear Model for XOR of Two Arbiter PUFs

Consider two k-stage Arbiter PUFs with their individual linear models:

$$\begin{aligned}\text{PUF}_1: \quad \Delta^{(1)} &= \mathbf{w}_1^T \mathbf{x} + b_1 \\ \text{PUF}_2: \quad \Delta^{(2)} &= \mathbf{w}_2^T \mathbf{x} + b_2\end{aligned}$$

where $\mathbf{x}$ is the common transformed challenge vector with components $x_i = \prod_{j=i}^{k-1} d_j$.

Individual Responses

The responses of each PUF are:

$$\begin{aligned}r_1 &= \frac{1 + \text{sign}(\Delta^{(1)})}{2} = \frac{1 + \text{sign}(\mathbf{w}_1^T \mathbf{x} + b_1)}{2} \\ r_2 &= \frac{1 + \text{sign}(\Delta^{(2)})}{2} = \frac{1 + \text{sign}(\mathbf{w}_2^T \mathbf{x} + b_2)}{2}\end{aligned}$$

XOR Response

The final response is the XOR of individual responses:

$$r = r_1 \oplus r_2$$

Linear Model Formulation

We can express the XOR challenge as a linear model in an extended feature space by first mapping the input vector $\mathbf{x}$ to a higher-dimensional space using the Kronecker product, resulting in the extended feature vector $\tilde{\mathbf{x}} = \mathbf{x} \otimes \mathbf{x}$. The XOR response r can then be expressed as a function of the signs of two linear projections of $\mathbf{x}$, namely

$$r = \frac{1 + \text{sign}(\Delta^{(1)}\Delta^{(2)})}{2} = \frac{1 + \text{sign}\left((\mathbf{w}_1^\top \mathbf{x} + b_1)(\mathbf{w}_2^\top \mathbf{x} + b_2)\right)}{2}.$$

Expanding this product yields

$$(\mathbf{w}_1^\top \mathbf{x} + b_1)(\mathbf{w}_2^\top \mathbf{x} + b_2) = \mathbf{x}^\top \mathbf{w}_1 \mathbf{w}_2^\top \mathbf{x} + b_2 \mathbf{w}_1^\top \mathbf{x} + b_1 \mathbf{w}_2^\top \mathbf{x} + b_1 b_2,$$

which can be written as a linear form in the extended space:

$$= \tilde{\mathbf{w}}^\top \tilde{\mathbf{x}} + \tilde{b},$$

where the weight vector in the lifted space is given by $\tilde{\mathbf{w}} = \text{vec}(\mathbf{w}_1 \mathbf{w}_2^\top)$ and the bias term by $\tilde{b} = b_1 b_2$. Here, $\tilde{\mathbf{x}}$ contains all second-order (quadratic) interaction terms of the original features, i.e., all $x_i x_j$.

Final Linear Model

Thus, in the extended space $\mathbb{R}^{k^2}$, there exists a linear model that can predict the XOR response as

$$r = \frac{1 + \text{sign}(\tilde{\mathbf{w}}^\top \tilde{\mathbf{x}} + \tilde{b})}{2}.$$

Khatri-Rao Product for Feature Map

This motivates the construction of a specific feature map $\tilde{\phi}(\mathbf{c})$, defined as the 2-fold Khatri-Rao product of a challenge vector $\mathbf{c} \in \{0, 1\}^8$ with itself:

$$\tilde{\phi}(\mathbf{c}) = \mathbf{c} \circ \mathbf{c} \in \mathbb{R}^{64},$$

where $\circ$ denotes the Khatri-Rao product. This operation explicitly models pairwise interactions between all input bits in $\mathbf{c}$, generating a 64-dimensional feature vector.

2) Dimensionality $\tilde{D}$ of the Feature Map

Given $\mathbf{c} \in \{0,1\}^8$, the dimensionality of the resulting feature map is simply

$$\tilde{D} = 8 \times 8 = 64.$$

Answer: The dimensionality of the feature map is $\tilde{D} = 64$.

Discussion

The 2-fold Khatri-Rao product is sufficient for capturing all pairwise interactions between the input features. If higher-order interactions are required, one could consider higher-order products. For example, a 4-fold Khatri-Rao product would yield

$$\tilde{D} = 8^4 = 4096,$$

but such expansions are generally unnecessary unless modeling very complex interactions.

3) Suitable Kernel when using Kernel SVM for solving ML-PUF problem

If the true feature mapping is $\tilde{\phi}(\mathbf{c}) = \mathbf{c} \circ \mathbf{c}$, then the kernel used in a kernel SVM should compute the inner product in this transformed space. That is,

$$\langle \tilde{\phi}(\mathbf{c}_1), \tilde{\phi}(\mathbf{c}_2) \rangle = \langle \mathbf{c}_1 \circ \mathbf{c}_1, \mathbf{c}_2 \circ \mathbf{c}_2 \rangle.$$

Given that the Khatri-Rao product corresponds to column-wise Kronecker product, this simplifies to

$$\langle \mathbf{c}_1 \circ \mathbf{c}_1, \mathbf{c}_2 \circ \mathbf{c}_2 \rangle = (\mathbf{c}_1^\top \mathbf{c}_2)^2.$$

Hence, the most appropriate kernel for this problem is a **degree-2 polynomial kernel**.

Recommended Kernel:

$$K(\mathbf{c}_1, \mathbf{c}_2) = (\gamma \mathbf{c}_1^\top \mathbf{c}_2 + r)^d \quad \text{with } d = 2.$$

Suggested Parameters:

- **Type:** Polynomial kernel
- **Degree:** $d = 2$
- **Coef:** $r = 0$ (no bias term)
- **Gamma:** $\gamma = 1$ (default scaling)

Justification:

- The 2-fold Khatri-Rao product implies quadratic feature interactions, matching a degree-2 polynomial kernel.
- RBF/Matern kernels are overly generic and fail to explicitly model pairwise interactions.
- Higher-degree kernels (e.g., $d = 4$) would overfit, as the true feature map is quadratic.

Conclusion: For perfect classification when $\tilde{\phi}(\mathbf{c}) = \mathbf{c} \circ \mathbf{c}$, use a **polynomial kernel of degree 2** with $\gamma = 1$ and $r = 0$.

4) Method for Delay Recovery

The forward model computes y from delays x via:

$$y = Ax$$

where $A \in \mathbb{R}^{65\times 256}$ encodes the PUF's linear relationships.

- For weight w_0:
$$w_0 = \frac{1}{2}(p_0 - q_0 + r_0 - s_0)$$
- For weights w_1 to w_{63}:
$$w_i = \frac{1}{2}\left[(p_i - q_i + r_i - s_i) + (p_{i-1} - q_{i-1} - r_{i-1} + s_{i-1})\right], \quad \text{for } i = 1 \text{ to } 63$$
- For bias b:
$$b = \frac{1}{2}(p_{63} - q_{63} - r_{63} + s_{63})$$

Optimization Problem

$$\begin{aligned} &\underset{x}{\text{minimize}} && \|Ax - y\|_2^2 \\ &\text{subject to} && x \geq 0 \quad \text{(element-wise non-negativity)} \end{aligned}$$

'A' Matrix Construction Details

Before the construction of 'A', consider a mapping. The function $\text{index}(\text{var}, i)$ is defined as:

$$\text{index}(\text{var}, i) = \text{offset}[\text{var}] + i$$

where offset is a mapping given by:

$$\text{offset} = \begin{cases} 'p' \mapsto 0, \\ 'q' \mapsto 64, \\ 'r' \mapsto 128, \\ 's' \mapsto 192. \end{cases}$$

Now,

- For $i = 0$:
$$A[0, \text{index}('p', 0)] = 0.5, \quad A[0, \text{index}('q', 0)] = -0.5,$$
$$A[0, \text{index}('r', 0)] = 0.5, \quad A[0, \text{index}('s', 0)] = -0.5,$$
- For $1 \leq i \leq 63$:
$$\begin{aligned} A[i, \text{index}('p', i)] &= 0.5 \\ A[i, \text{index}('q', i)] &= -0.5 \\ A[i, \text{index}('r', i)] &= 0.5 \\ A[i, \text{index}('s', i)] &= -0.5 \\ A[i, \text{index}('p', i-1)] &= 0.5 \\ A[i, \text{index}('q', i-1)] &= -0.5 \\ A[i, \text{index}('r', i-1)] &= -0.5 \\ A[i, \text{index}('s', i-1)] &= 0.5 \end{aligned}$$
- For bias (row 64):
$$A[64, \text{index}('p', 63)] = 0.5, \quad A[64, \text{index}('q', 63)] = -0.5,$$
$$A[64, \text{index}('r', 63)] = -0.5, \quad A[64, \text{index}('s', 63)] = 0.5,$$

Algorithm Implementation

For a given linear model vector $y \in \mathbb{R}^{65}$ (64 weights + 1 bias), recovered delays $p, q, r, s \in \mathbb{R}^{64}_{+}$, construct constraint matrix $A \in \mathbb{R}^{65\times256}$ via build_matrix_A(), then initialize solution vector $x^{(0)} \leftarrow \epsilon \cdot \mathbf{1}_{256}$ $\epsilon > 0$ (typically 10^{-6}), and then finally solve the constrained optimization problem:

$$x^* = \arg\min_{x\in\mathbb{R}^{256}} \quad \|Ax - y\|_2^2$$

$$\text{subject to} \quad x \geq 0 \quad \text{(element-wise non-negativity)}$$

$$p = x^*_{0:63}, \qquad q = x^*_{64:127},$$
$$r = x^*_{128:191}, \quad s = x^*_{192:255}$$

5) & 6) Coding the ML-PUF Problem and Arbiter PUF Inversion Problem to Recover Delays (Code in submit.py)

7) Experiments with LogisticRegression and LinearSVC

We evaluate hyperparameter impacts using both classifiers. Summary below:

Model	C	Penalty	Loss	Train Time (s)	Accuracy (%)
LogisticRegression	1.0	l1	–	0.6202	93.56
LogisticRegression	1.0	l2	–	0.1111	92.19
LinearSVC	1.0	l1	squared hinge	1.1320	93.00
LinearSVC	1.0	l2	squared hinge	0.0384	91.63
LogisticRegression	0.01	l2	–	0.0295	74.62
LogisticRegression	1.0	l2	–	0.1111	92.19
LogisticRegression	100	l2	–	0.0905	96.13
LogisticRegression	140	l2	–	0.0903	97.00
LinearSVC	0.01	l2	squared hinge	0.0244	86.44
LinearSVC	1.0	l2	squared hinge	0.0384	91.63
LinearSVC	100	l2	squared hinge	0.0435	93.37

Table 1: Effect of regularization (penalty) and C (Hyperparameter) on performance for LogisticRegression and LinearSVC

Model	C	Penalty	tol	Train Time (s)	Accuracy (%)
LinearSVC	1.0	l2	10^{-6}	0.0859	91.63
LinearSVC	1.0	l2	10^{-4}	0.0535	91.63
LinearSVC	1.0	l2	10^{-2}	0.0414	92.50

Table 2: Effect of tol on performance for LinearSVC with squared hinge loss

Inferences:

- **Effect of C:** Higher C (weaker regularization) improves accuracy for both models. For LogisticRegression, accuracy increases from 74.62% at $C = 0.01$ to 97.00% at $C = 140$, indicating that weaker regularization can help reduce overfitting. A similar trend is observed in LinearSVC, where accuracy improves from 86.44% at $C = 0.01$ to 93.37% at $C = 100$.
- **L1 vs L2:** The L2 penalty tends to offer more stability and slightly better results overall. LogisticRegression with L2 achieves accuracy values of 92.19% at $C = 1.0$ and 96.13% at $C = 100$, while L1 achieves a slightly higher accuracy (93.56%) at $C = 1.0$. However, for LinearSVC, L1 results in higher training times (1.1320s) and less stable performance compared to L2.

- *tol* **in LinearSVC:** Increasing the tolerance (tol) from 10^{-6} to 10^{-2} in LinearSVC significantly reduced training time while slightly improving accuracy, indicating that a higher tolerance can enhance efficiency without compromising performance.
- **Model Comparison:** LogisticRegression with L2 delivers better performance, achieving a higher accuracy (96.13% at $C = 100$) compared to LinearSVC, which reaches only 93.37% at $C = 100$. Moreover, LogisticRegression trains faster at these accuracy levels, making it a more efficient choice for practical applications.
- **Efficiency:** LinearSVC with L1 has considerably higher training times (1.1320s) and a lower accuracy (93.00%) compared to other configurations. This suggests that LinearSVC with L1 should only be used when sparsity is essential, but it may be less efficient than LogisticRegression or LinearSVC with L2, even in those cases.