CS771 : Introduction to Machine Learning

Assignment - 1

Team: NEURAL KNIGHTS

Nitika Singh
220732
Dept. of Materials Science and Engineering
IIT Kanpur
nitikas22@iitk.ac.in

Pranjali Singh
220796
Dept. of Electrical Engineering
IIT Kanpur
pranjalis22@iitk.ac.in

Ragha Shrutilaya M
220850
Dept. of Electrical Engineering
IIT Kanpur
raghasm22@iitk.ac.in

Palak Agrawal
220757
Dept. of Electrical Engineering
IIT Kanpur
apalak22@iitk.ac.in

Pubali Banerjee
220192
Dept. of Electrical Engineering
IIT Kanpur
pubalib22@iitk.ac.in

Riddhima Vijayvargiya
220883
Dept. of Electrical Engineering
IIT Kanpur
riddhima22@iitk.ac.in

Abstract:
This report presents our solution to the Mini Project of CS771, where we investigate the vulnerability of a Companion Arbiter PUF to machine learning attacks. We mathematically derive how a single linear model can predict the responses of the ML-PUF by exploiting delay-based correlations between its component PUFs. We then evaluate the performance of our model, highlighting the impact of hyperparameter tuning on training time and prediction accuracy.

1 Mathematical Derivation of Linear Model for Predicting ML-PUF Responses

We will use a straightforward approach to solve this problem. The strategy involves leveraging the known differences in time delays for the upper and lower signals, as well as the sum of their delays. First, we will derive the difference in time delays for 8-bit input in a way similar to as outlined in the lectures, and then proceed to derive the sum of these delays. Also the notations are same as described in lectures.

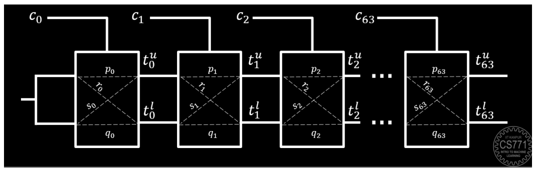


Figure 1: Sample figure caption.

1.1 Deriving the difference of Delays

Here, t_i^u and t_i^l represent the times at which the signal departs from the i^{th} MUX pair. Expressing t_i^u in terms of t_{i-1}^u, t_{i-1}^l and c_i, we obtain:

$$t_i^u = (1 - c_i) \cdot (t_{i-1}^u + p_i) + c_i \cdot (t_{i-1}^l + s_i) \tag{1}$$
$$t_i^l = (1 - c_i) \cdot (t_{i-1}^l + q_i) + c_i \cdot (t_{i-1}^u + r_i) \tag{2}$$

Thus, following the lecture slides we have, $\Delta_i = t_i^u - t_i^l$. Subtracting equations (1), (2) we get, $\Delta_i = \Delta_{i-1} \cdot d_i + \alpha_i \cdot d_i + \beta_i$, where α_i, β_i depend on constants that are indeterminable from the physical/measurable perspective and thus we call them system constants and are given by:

$$\alpha_i = \frac{(p_i - q_i + r_i - s_i)}{2}, \quad \beta_i = \frac{(p_i - q_i - r_i + s_i)}{2} + a$$

where p_i, q_i, r_i, s_i are system parameters. Also d_i is governed by the challenge bits/input (c_i's):

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

$$\Delta_{-1} = 0$$

Moreover, observing the recursion unfold carefully we can simplify this relation further:

$$\begin{aligned}
\Delta_0 &= \alpha_0 \cdot d_0 + \beta_0 \\
\Delta_1 &= \alpha_0 \cdot d_1 \cdot d_0 + (\alpha_1 + \beta_0) \cdot d_1 + \beta_1 \\
\Delta_2 &= \alpha_0 \cdot d_2 \cdot d_1 \cdot d_0 + (\alpha_1 + \beta_0) \cdot d_2 \cdot d_1 + (\alpha_2 + \beta_1) \cdot d_2 + \beta_2 \\
&\vdots
\end{aligned}$$

The only Δ we require is Δ_7, the last output.

$$\Delta_7 = w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_7 \cdot x_7 + \beta_7 = \mathbf{w}^T \cdot \mathbf{x} + b = \mathbf{W}^T \cdot \mathbf{X}$$

where $\mathbf{w}, \mathbf{x}$ are 8-dimensional vectors and $\mathbf{W}, \mathbf{X}$ are 9-dimensional, just including the bias term $W_8 = \beta_7$ and $X_8 = 1$. Each term of $\mathbf{w}, \mathbf{x}$ being:

$$b = \beta_7, \quad w_0 = \alpha_0, \quad w_i = \alpha_i + \beta_{i-1}, \quad x_i = \prod_{j=i}^{7} d_j$$

1.2 Deriving the Sum of Delays

We have,

$$t_i^u = (1 - c_i) \cdot (t_{i-1}^u + p_i) + c_i \cdot (t_{i-1}^l + s_i) \tag{1}$$
$$t_i^l = (1 - c_i) \cdot (t_{i-1}^l + q_i) + c_i \cdot (t_{i-1}^u + r_i) \tag{2}$$

Adding both the equations,

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

Put

$$(s_i + r_i - p_i - q_i) = r_i'$$

to get,

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

Substituting the value of $i = 7, 6, \ldots, 0$ and adding the equations:

$$\begin{aligned} t_7^u + t_7^l = {} & (c_7 \cdot r_7') + (p_7 + q_7) + \\ & (c_6 \cdot r_6') + (p_6 + q_6) + \\ & \cdots + \\ & (c_0 \cdot r_0') + (p_0 + q_0) \end{aligned}$$

We finally get,

$$t_7^u + t_7^l = \sum_{i=0}^{7} (c_i \cdot r_i') + \sum_{i=0}^{7} (p_i + q_i) \tag{1}$$

where

$$r_i' = s_i + r_i - p_i - q_i$$

We also derived the difference in previous sections as:

$$t_7^u - t_7^l = w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_7 \cdot x_7 + \beta_7 \tag{2}$$

where

$$w_0 = \alpha_0, \quad w_i = \alpha_i + \beta_{i-1}, \quad x_i = \prod_{j=i}^{7} d_j, \quad d_i = (1 - 2 \cdot c_i)$$

and

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

Adding (1) and (2):

$$2 \cdot t_7^u = (c_7 \cdot r_7') + (c_6 \cdot r_6') + \cdots + (c_0 \cdot r_0') \\ + w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_7 \cdot x_7 + \beta_7$$

Dividing both sides by 2:

$$t_7^u = \frac{1}{2} \left(\sum_{i=0}^{7} c_i \cdot r_i' + \sum_{i=0}^{7} w_i \cdot x_i + \beta_7 \right)$$

Rewriting $c_7 = 1 - 2c_7$ and taking c_7 common from $c_7 \cdot r_7'$ and $2 \cdot w_7 \cdot c_7$, we get:

$$t_7^u = \frac{1}{2} \left(c_7 \cdot (r_7' - 2w_7) + \sum_{i=0}^{6} c_i \cdot r_i' + \sum_{i=0}^{7} w_i \cdot x_i + \beta_7 + w_7 \right)$$

In matrix form, this can be expressed as:

$$t_7^u = \frac{1}{2} \left(\mathbf{W}^T \cdot \phi(c) + b \right)$$

where:

$$\phi(c) = [c_7, c_6, \ldots, c_0, x_0, x_1, \ldots, x_6]^T$$

$$\mathbf{W} = [r_7' - 2w_7, r_6', \ldots, r_0', w_0, w_1, \ldots, w_6]^T, \quad b = \beta_7 + w_7$$

In this solution we denote parameters for PUF0 as $A_{0,i}$, and parameters for PUF1 as $A_{1,i}$.

Linear Model for Response0:

$$t_{0,7}^u = \frac{1}{2} \left((c_7 \cdot (r_{0,7} - 2w_{0,7})) + (c_6 \cdot r_{0,6}) + \cdots + (c_0 \cdot r_{0,0}) + w_{0,0} \cdot x_0 + \cdots + w_{0,7} \cdot x_7 + \beta_{0,7} \right)$$

$$t_{1,7}^u = \frac{1}{2} \left((c_7 \cdot (r_{1,7} - 2w_{1,7})) + (c_6 \cdot r_{1,6}) + \cdots + (c_0 \cdot r_{1,0}) + w_{1,0} \cdot x_0 + \cdots + w_{1,7} \cdot x_7 + \beta_{1,7} \right)$$

Now

$$t_{0,7}^u - t_{1,7}^u = \frac{1}{2}\Big(c_7 \cdot ((r_{0,7} - 2w_{0,7}) - (r_{1,7} - 2w_{1,7})) + \\ c_6 \cdot (r_{0,6} - r_{1,6}) + \cdots + c_0 \cdot (r_{0,0} - r_{1,0}) + \\ (w_{0,0} - w_{1,0}) \cdot x_0 + \cdots + (w_{0,6} - w_{1,6}) \cdot x_6 + \\ (\beta_{0,7} + w_{0,7}) - (\beta_{1,7} + w_{1,7}) \Big)$$

In matrix form, this can be expressed as:

$$t_{0,7}^u - t_{1,7}^u = \frac{1}{2} \left(\tilde{W}^T \cdot \phi(c) + \tilde{b} \right)$$

Where:

$$\phi(c) = [c_7, c_6, \ldots, c_0, x_0, x_1, \ldots, x_6]^T$$

$$\tilde{W} = [(r_{0,7} - 2w_{0,7}) - (r_{1,7} - 2w_{1,7}),\ r_{0,6} - r_{1,6}, \ldots, r_{0,0} - r_{1,0},\ w_{0,0} - w_{1,0}, \ldots, w_{0,6} - w_{1,6}]^T$$

$$\tilde{b} = (\beta_{0,7} + w_{0,7}) - (\beta_{1,7} + w_{1,7})$$

Linear Model for Response1:

$$t^l_{0,7} = \frac{1}{2}\left((c_7 \cdot (r_{0,7} + 2w_{0,7})) + \cdots + (c_0 \cdot r_{0,0}) - w_{0,0} \cdot x_0 - \cdots - w_{0,7} \cdot x_7 - (\beta_{0,7} + w_{0,7})\right)$$

$$t^l_{1,7} = \frac{1}{2}\left((c_7 \cdot (r_{1,7} + 2w_{1,7})) + \cdots + (c_0 \cdot r_{1,0}) - w_{1,0} \cdot x_0 - \cdots - w_{1,7} \cdot x_7 - (\beta_{1,7} + w_{1,7})\right)$$

Now

$$\begin{aligned} t^l_{0,7} - t^l_{1,7} = \frac{1}{2}\Big(& c_7 \cdot ((r_{0,7} + 2w_{0,7}) - (r_{1,7} + 2w_{1,7})) + \\ & c_6 \cdot (r_{0,6} - r_{1,6}) + \cdots + c_0 \cdot (r_{0,0} - r_{1,0}) - \\ & (w_{0,0} - w_{1,0}) \cdot x_0 - \cdots - (w_{0,6} - w_{1,6}) \cdot x_6 - \\ & ((\beta_{0,7} + w_{0,7}) - (\beta_{1,7} + w_{1,7}))\Big) \end{aligned}$$

In matrix form, this can be expressed as:

$$t^l_{0,7} - t^l_{1,7} = \frac{1}{2}\left(\tilde{W}^T \cdot \phi(c) + \tilde{b}\right)$$

Where:

$$\phi(c) = [c_7, c_6, \ldots, c_0, x_0, x_1, \ldots, x_6]^T$$

$$\tilde{W} = [(r_{0,7} + 2w_{0,7}) - (r_{1,7} + 2w_{1,7}),\ r_{0,6} - r_{1,6}, \ldots, r_{0,0} - r_{1,0},\ -(w_{0,0} - w_{1,0}), \ldots, -(w_{0,6} - w_{1,6})]^T$$

$$\tilde{b} = (\beta_{1,7} + w_{1,7}) - (\beta_{0,7} + w_{0,7})$$

We now compute the multiplication of the two expressions to model the XOR response of the ML-PUF, as per MP1.

Recall the expressions:

$$\begin{aligned} t^u_{0,7} - t^u_{1,7} = \frac{1}{2}\big(& c_7 \cdot ((r_{0,7} - 2w_{0,7}) - (r_{1,7} - 2w_{1,7})) + c_6 \cdot (r_{0,6} - r_{1,6}) + \cdots + c_0 \cdot (r_{0,0} - r_{1,0}) \\ & + (w_{0,0} - w_{1,0}) \cdot x_0 + \cdots + (w_{0,6} - w_{1,6}) \cdot x_6 + (\beta_{0,7} + w_{0,7}) - (\beta_{1,7} + w_{1,7})\big) \end{aligned}$$

$$\begin{aligned} t^l_{0,7} - t^l_{1,7} = \frac{1}{2}\big(& c_7 \cdot ((r_{0,7} + 2w_{0,7}) - (r_{1,7} + 2w_{1,7})) + c_6 \cdot (r_{0,6} - r_{1,6}) + \cdots + c_0 \cdot (r_{0,0} - r_{1,0}) \\ & - (w_{0,0} - w_{1,0}) \cdot x_0 - \cdots - (w_{0,6} - w_{1,6}) \cdot x_6 - ((\beta_{0,7} + w_{0,7}) - (\beta_{1,7} + w_{1,7}))\big) \end{aligned}$$

Now multiplying:

$$(t^u_{0,7} - t^u_{1,7}) \cdot (t^l_{0,7} - t^l_{1,7}) = \frac{1}{4} \cdot \Big(\text{Expression}_u\Big) \cdot \Big(\text{Expression}_l\Big)$$

Writing all terms symbolically:

$$\begin{aligned} (t^u_{0,7} - t^u_{1,7}) \cdot (t^l_{0,7} - t^l_{1,7}) = \frac{1}{4}\Bigg[& \left(\sum_{i=0}^{7} c_i \cdot \Delta r_i^{(u)} + \sum_{j=0}^{6} \Delta w_j^{(u)} \cdot x_j + b_u\right) \cdot \\ & \left(\sum_{i=0}^{7} c_i \cdot \Delta r_i^{(l)} - \sum_{j=0}^{6} \Delta w_j^{(u)} \cdot x_j - b_u\right)\Bigg] \end{aligned}$$

Where:

- $\Delta r_7^{(u)} = (r_{0,7} - 2w_{0,7}) - (r_{1,7} - 2w_{1,7})$ - $\Delta r_7^{(l)} = (r_{0,7} + 2w_{0,7}) - (r_{1,7} + 2w_{1,7})$ - $\Delta r_i^{(u)} = r_{0,i} - r_{1,i}, \quad \text{for } i = 0 \text{ to } 6$ - $\Delta r_i^{(l)} = r_{0,i} - r_{1,i}, \quad \text{for } i = 0 \text{ to } 6$ - $\Delta w_j^{(u)} = w_{0,j} - w_{1,j}$ - $b_u = (\beta_{0,7} + w_{0,7}) - (\beta_{1,7} + w_{1,7})$

This final expression models the XOR logic by taking the product of the two linear expressions.

To obtain the predicted response bit, we apply:

$$\boxed{\text{Response}(c) = \frac{1 - \text{sign}\left((t^u_{0,7} - t^u_{1,7}) \cdot (t^l_{0,7} - t^l_{1,7})\right)}{2}}$$

Therefore, the final feature map is:

$$\begin{aligned}
&(c_1c_1,\ c_1c_2, \ldots, c_7c_6,\ c_7c_7,\\
&\ x_1x_1,\ x_1x_2, \ldots, x_6x_5,\ x_6x_6,\\
&\ c_1x_1,\ c_1x_2, \ldots, c_7x_5,\ c_7x_6,\\
&\ x_1c_1,\ x_1c_2, \ldots, x_6c_6,\ x_6c_7,\\
&\ c_1.1,\ c_2.1, \ldots, c_7.1,\ x_1.1, \ldots, x_6.1,\\
&\ 1.c_1,\ 1.c_2, \ldots, 1.c_7,\ 1.x_1, \ldots, 1.x_6, 1)
\end{aligned}$$

2 Dimensionality $\tilde{D}$ Required by the Linear Model for ML-PUF Prediction

2.1 Initial Feature Construction

We begin by identifying the set of base variables involved in forming the feature space. These include:

- 8 challenge bits: $c_7, c_6, \ldots, c_0$
- 7 intermediate state bits: $x_0, x_1, \ldots, x_6$
- Constant term: 1

This gives us a total of $8 + 7 + 1 = 16$ base variables.

We construct all second-order pairwise product features from these variables. This includes terms of the form:

$$u_i \cdot u_j \quad \text{where } u_i, u_j \in \{c_7, \ldots, c_0, x_0, \ldots, x_6, 1\}$$

Naively, this yields:

$$16 \times 16 = 256 \text{ features}$$

2.2 Removing Duplicate Terms (Due to Commutativity)

Because multiplication is commutative, $u_iu_j = u_ju_i$, we remove duplicate terms such as c_7c_6 and c_6c_7. Hence, we consider unordered pairs (including self-pairs). The number of such combinations is:

$$\binom{16+1}{2} = \binom{17}{2} = 136$$

2.3 Removing Redundant Self-Terms for x_i

Each x_i is defined as:

$$x_i = \prod_{k=i}^{7} d_k, \quad \text{where } d_k = 1 - 2c_k \in \{-1, +1\}$$

This implies:

$$x_i^2 = 1 \quad \text{for all } i$$

Therefore, all terms of the form $x_i \cdot x_i$ are constant and provide no useful information for the model. We remove the 7 such terms:

$$x_0x_0, x_1x_1, \ldots, x_6x_6$$

Note: 1 should also be removed

2.4 Removing Redundant Self-Terms for c_i

Since each challenge bit $c_i \in \{0, 1\}$, we have $c_i^2 = c_i$ for all i. Therefore, quadratic terms of the form $c_i \cdot c_i$ do not contribute any new information and can be safely removed from the feature map. We eliminate these 8 redundant terms:

$$c_0c_0,\ c_1c_1,\ \ldots,\ c_7c_7$$

2.5 Removing Redundant x_ix_{i+1} and x_ix_{i+2} Terms

Each intermediate variable x_i is defined as:

$$x_i = \prod_{k=i}^{7} d_k, \quad \text{where } d_k = 1 - 2c_k \in \{-1, +1\}$$

This implies that certain pairwise products of x_i variables can be exactly represented using combinations of c_i bits:

- $$\begin{aligned} x_ix_{i+1} &= \left(\prod_{k=i}^{7} d_k\right)\left(\prod_{k=i+1}^{7} d_k\right) \\ &= d_i \cdot \left(\prod_{k=i+1}^{7} d_k\right)^2 \\ &= d_i \cdot (x_{i+1})^2 \end{aligned}$$

  Since $x_{i+1} = \pm 1$, we know that $(x_{i+1})^2 = 1$ always.
  Now, $d_i = 1 - 2c_i$, which is a linear combination of c_i, and c_i is already part of the feature map.
  Therefore, all six x_ix_{i+1} terms can be removed.

- $$\begin{aligned} x_ix_{i+2} &= \left(\prod_{k=i}^{7} d_k\right)\left(\prod_{k=i+2}^{7} d_k\right) \\ &= d_id_{i+1} \cdot \left(\prod_{k=i+2}^{7} d_k\right)^2 \\ &= d_id_{i+1} \cdot (x_{i+2})^2 \end{aligned}$$

  Since $x_{i+2} = \pm 1$, we have $(x_{i+2})^2 = 1$ always.
  Now, $d_id_{i+1} = (1 - 2c_i)(1 - 2c_{i+1})$, which is a linear combination of terms already present in the feature map.
  Therefore, all five x_ix_{i+2} terms can be removed.

Hence, these terms do not contribute new information beyond what is already captured by first-order or product terms of the challenge bits. Consequently, we remove all such redundant terms of the form x_ix_{i+1} and x_ix_{i+2} from the feature set, resulting in a total of 11 terms removed.

2.6 Removing Redundant c_ix_{i+1} Terms

We analyze terms of the form c_ix_{i+1} in the feature space, where:

$$x_i = \prod_{k=i}^{7} d_k = \prod_{k=i}^{7} (1 - 2c_k)$$

To simplify c_ix_{i+1}, we start by writing:

$$x_i = d_ix_{i+1} = (1 - 2c_i)x_{i+1} \Rightarrow c_ix_{i+1} = \frac{1}{2}(x_{i+1} - x_i)$$

This identity implies that c_ix_{i+1} is a linear combination of x_i and x_{i+1}, both of which are already included in the feature space.

Therefore, terms of the form c_ix_{i+1} (6 terms) do not provide new independent information and can be safely removed from the feature set.

2.7 Reducing x_6 Term as a Linear Combination

The intermediate state bit x_i is defined as:

$$x_i = \prod_{k=i}^{7}(1-2c_k) = \sum_{S\subseteq\{i,\ldots,7\}} (-2)^{|S|} \prod_{k\in S} c_k$$

For $i = 6$:

$$x_6 = (1-2c_6)(1-2c_7) = 1-2c_6-2c_7+4c_6c_7$$

Terms: constant (1), c_6, c_7, c_6c_7, all present in the feature map (c_i, c_ic_j for $i<j$, constant term). Thus:

$$x_6 = 1-2c_6-2c_7+4c_6c_7$$

x_6 can be removed as it is exactly expressed using terms in the feature map.

Reducing c_6x_6 and c_7x_6 Terms

To determine if c_6x_6 and c_7x_6 can be reduced, we check if they can be exactly expressed as linear combinations of the feature map's basis: $\{c_ic_j \mid 0\le i<j\le 7\} \cup \{c_i \mid 0\le i\le 7\}$ ($28+8=36$ terms). The intermediate state bit is:

$$\begin{aligned} x_6 &= \prod_{k=6}^{7} d_k = d_6d_7 = (1-2c_6)(1-2c_7) \\ &= 1-2c_6-2c_7+4c_6c_7 \end{aligned}$$

For c_6x_6

$$\begin{aligned} c_6x_6 &= c_6(1-2c_6-2c_7+4c_6c_7) \\ &= c_6-2c_6c_6-2c_6c_7+4c_6c_6c_7 \end{aligned}$$

Since $c_6^2 = c_6$, $c_6^2c_7 = c_6c_7$:

$$\begin{aligned} c_6x_6 &= c_6-2c_6-2c_6c_7+4c_6c_7 \\ &= -c_6+2c_6c_7 \end{aligned}$$

Terms c_6 and c_6c_7 are in the basis. Thus, c_6x_6 is redundant:

$$c_6x_6 = -1\cdot c_6 + 2\cdot c_6c_7$$

For c_7x_6

$$\begin{aligned} c_7x_6 &= c_7(1-2c_6-2c_7+4c_6c_7) \\ &= c_7-2c_7c_6-2c_7c_7+4c_7c_6c_7 \end{aligned}$$

Since $c_7^2 = c_7$, $c_7^2c_6 = c_7c_6$:

$$\begin{aligned} c_7x_6 &= c_7-2c_6c_7-2c_7+4c_6c_7 \\ &= -c_7+2c_6c_7 \end{aligned}$$

Terms c_7 and c_6c_7 are in the basis. Thus, c_7x_6 is redundant:

$$c_7x_6 = -1\cdot c_7 + 2\cdot c_6c_7$$

Both c_6x_6 and c_7x_6 are redundant and can be removed, as they are exactly expressible using c_i and c_ic_j, therefore 2 terms being removed.

D̃ = 136-8-8-11-6-1-2=100

3 Kernel SVM for ML-PUF Classification

The components of $\tilde{\phi}(c)$ can be viewed as monomials involving $d_0, d_1, \ldots, d_{n-1}$. Given that $d_i = 1-2c_i$, these expressions can be reformulated entirely in terms of $c_0, c_1, \ldots, c_{n-1}$. Therefore, a feature transformation comprising all monomials of these challenge bits is sufficient for our representation.

To compute the inner product in this expanded feature space without explicitly constructing it, we employ the **polynomial kernel**, defined as:

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + r)^d$$

This kernel function effectively captures a combination of monomials of degrees ranging from 0 up to d, aligning with the structure we aim to represent.

- **Degree:** $d = n$, to ensure inclusion of all monomials up to degree n.
- **Offset:** $r = 1$, to include lower-degree monomials and the constant term.

To achieve perfect classification, the SVM kernel must implicitly generate a feature space equivalent to the 100-dimensional explicit map, capturing all monomials required by the ML-PUF response.

Polynomial Kernel Details

Consider a polynomial kernel of degree d:

$$K(c, c') = (\langle c, c' \rangle + r)^d \tag{1}$$

where:

- $c, c' \in \{0,1\}^8$
- Inner product: $\langle c, c' \rangle = \sum_{i=0}^{7} c_i c'_i$ (note: $c_i^2 = c_i$ in binary)

This induces a feature space that includes all monomials of the challenge bits c_i up to degree d.

Feature Space Size

To capture all required terms including interactions like x_j = monomials in c_i, we need a space with up to degree-9 monomials:

This space covers:

- Linear terms: c_i (degree 1)
- Quadratic terms: $c_i c_j$
- Higher-order terms: $c_i x_j, x_i x_j$ up to degree 8 or approximated through combinations

Kernel Parameters

We choose:

- **Degree:** $d = 9$, to capture all monomials up to x_0
- **Offset:** $r = 1$, to ensure inclusion of constant terms
- **Gamma:** Default ($\gamma = 1/9$), not critical for polynomial kernels

Conclusion

The polynomial kernel:

$$K(c, c') = (c^\top c' + 1)^9$$

generates a 256-dimensional feature space, sufficient to represent the 100-dimensional explicit map used for ML-PUF classification. It ensures all required polynomial interactions are represented, leading to perfect classification.

Alternative kernels like RBF and Matern are less suitable, as they do not target the specific polynomial structure inherent in the ML-PUF design.

4 Delay Recovery by Inverting an Arbiter PUF Model

We aim to recover 256 non-negative stage delays $x_{\text{delays}} \in \mathbb{R}^{256}_{\geq 0}$ that yield a given 65-dimensional Arbiter PUF linear model $y = (w_0, w_1, \ldots, w_{63}, b)^T \in \mathbb{R}^{65}$.

4.1 Model Structure

Let:

$$x_{\text{delays}} = (p_0, q_0, r_0, s_0, \dots, p_{63}, q_{63}, r_{63}, s_{63})^\top \in \mathbb{R}^{256}$$

We define the following intermediate terms:

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

The output values are formed by:

$$w_0 = \alpha_0, \quad w_i = \alpha_i + \beta_{i-1}\ (1 \leq i \leq 63), \quad b = \beta_{63}$$

4.2 Step 1: Forward Model

We construct the sparse linear system $Ax_{\text{delays}} = y$, where $A \in \mathbb{R}^{65\times 256}$, $x_{\text{delays}} = (p_0, q_0, r_0, s_0, \dots, p_{63}, q_{63}, r_{63}, s_{63})^T$. The model is defined as follows (for $N = 64$ stages, 0–63):

1. **Intermediate delays:** $\alpha_i = \frac{1}{2}(p_i - q_i + r_i - s_i)$, $\beta_i = \frac{1}{2}(p_i - q_i - r_i + s_i)$ for $i = 0, \dots, 63$.
2. **Model parameters:** $w_0 = \alpha_0$, $w_i = \alpha_i + \beta_{i-1}$ for $1 \leq i \leq 63$, and $b = \beta_{63}$.
3. **Coefficient matrix A:** The system $Ax_{\text{delays}} = y$ encodes these relationships. A is sparse, with at most 8 non-zeros per row. Partitioning columns into 64 blocks of 4 (for p_i, q_i, r_i, s_i), the structure is:

$$A = \begin{pmatrix} A^{(0)} & 0 & 0 & \cdots & 0 \\ B^{(1)} & A^{(1)} & 0 & \cdots & 0 \\ 0 & B^{(2)} & A^{(2)} & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & B^{(63)} & A^{(63)} \\ 0 & \cdots & 0 & 0 & B^{(64)} \end{pmatrix} \in \mathbb{R}^{65\times 256}$$

where the non-zero blocks are:

- **Row 0 ($w_0 = \alpha_0$):** Block $A^{(0)} = \left[\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}\right]$ in column block 0.
- **Rows $r = 1 \dots 63$ ($w_r = \alpha_r + \beta_{r-1}$):** Block $B^{(r)} = \left[\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}\right]$ (for β_{r-1}) in column block $r-1$, and block $A^{(r)} = \left[\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}\right]$ (for α_r) in column block r.
- **Row 64 ($b = \beta_{63}$):** Block $B^{(64)} = \left[\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2}, \frac{1}{2}\right]$ in column block 63.

Sparsity summary: Rows 0 and 64 have 4 non-zeros; rows 1 to 63 have 8 non-zeros.

4.2 Step 2: Inversion under Non-Negativity

To recover a physically valid solution ($x_{\text{delays}} \geq 0$), we solve the Non-negative Least Squares (NNLS) problem:

$$\min_{x \in \mathbb{R}^{256}_{\geq 0}} \|Ax - y\|_2^2 \quad \text{subject to } x \geq 0.$$

This can be solved using custom sparse solvers like projected gradient or coordinate descent, adapted for the non-negativity constraint $x \geq 0$. This yields a non-negative delay vector x_{delays} that best reproduces the model y.

5 ML-PUF Learning: Linear Model Training Using Sklearn Classifiers Code

Code Submitted

6 Arbiter PUF Inversion: Delay Recovery Using Linear Solvers Code

Code Submitted

7 Experimental Results: Hyperparameter Analysis of LinearSVC and LogisticRegression for ML-PUF Learning

7.1 Modifying the `loss` Hyperparameter in LinearSVC (Hinge vs. Squared Hinge)

The `loss` function determines how the model assesses errors in predictions during training. The `LinearSVC` class from `sklearn.svm` supports two loss functions: `hinge` and `squared_hinge`.

7.1.1 Setup

- $C = 100$, `tolerance` $= 1e{-}3$, `penalty` $=$ l2, `max_iter` $= 5000$

Table 1: Impact of Changing Loss Hyperparameters in LinearSVC

Loss	**Training Time (s)**	**Accuracy**
Hinge	0.6708	1.000
Squared Hinge	0.9640	1.000

7.2 Tuning the C Hyperparameter in LinearSVC and LogisticRegression

The regularization strength C influences model complexity. Lower C enforces stronger regularization, promoting generalization but potentially underfitting. Higher C allows more complex models, which may fit training data more closely but risk overfitting.

Configurations Used

- `LinearSVC: tol=1e-3, penalty='l2', max_iter=5000, loss='hinge', dual=True`
- `LogisticRegression:  solver='lbfgs', max_iter=5000, tol=1e-3, penalty='l2'`

Table 2: Effect of Varying C in LinearSVC and LogisticRegression

Model	**C Value**	**Training Time (s)**	**Accuracy**
LinearSVC	1	0.2766	0.9962
	10	0.6722	1.000
	100	0.6527	1.000
LogisticRegression	1	0.0808	0.9888
	10	0.0902	1.000
	100	0.1900	1.000

7.3 Varying the `penalty` Hyperparameter in LinearSVC and LogisticRegression (L1 vs. L2)

The `penalty` parameter determines the type of regularization:

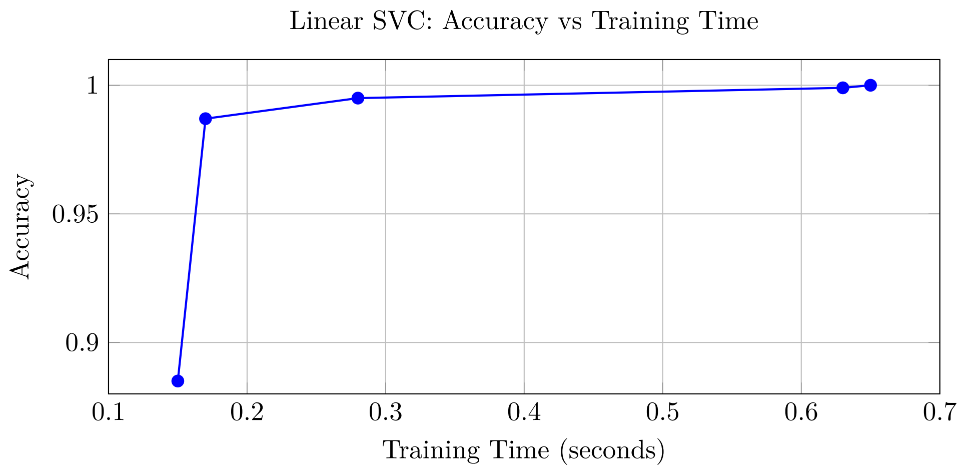
- **L2**: Encourages smaller coefficients by adding the square of weights to the loss.
- **L1**: Promotes sparsity by driving some coefficients exactly to zero.

Used Configuration

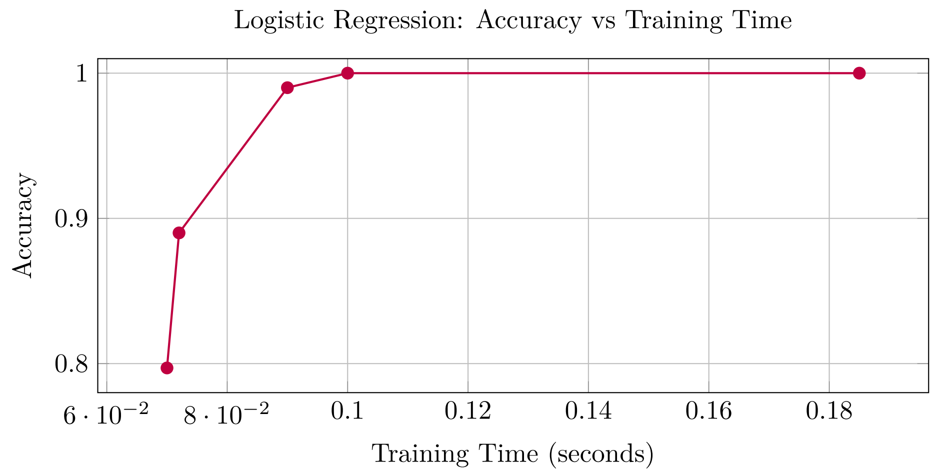
- `LinearSVC: C=100, loss='squared_hinge', tol=1e-3, dual=False, max_iter=5000`
- `LogisticRegression:  C=100, solver='lbfgs', tol=1e-3, max_iter=5000`

Note: In `LinearSVC`, the acceptable combinations of `penalty` and `dual` are:

- For `penalty='l2'`: `dual=True`
- For `penalty='l1'`: `dual=False`



(a) Linear SVC

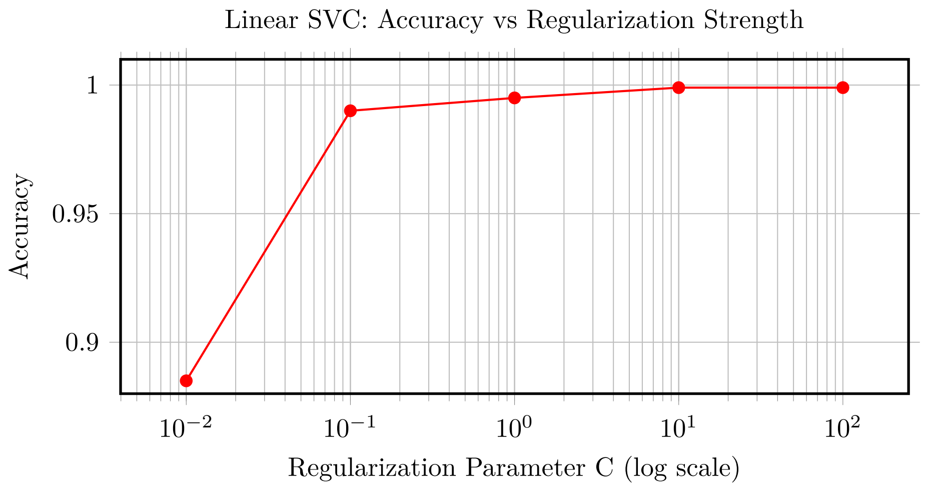


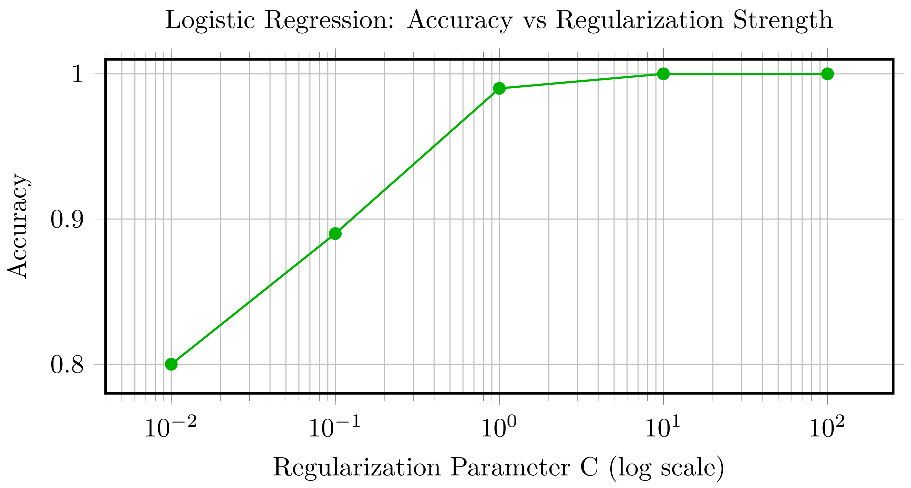
(b) Logistic Regression

Figure 2: Comparison of Accuracy vs Training Time for Linear SVC and Logistic Regression

Table 3: Effect of Penalty Type on Training Time and Accuracy

Model	**Penalty Type**	**Training Time (s)**	**Accuracy**
LinearSVC	L1	5.3132	1.000
LinearSVC	L2	0.9294	1.000
LogisticRegression	L1	0.8075	1.000
LogisticRegression	L2	0.1900	1.000


Linear SVC: Accuracy vs Regularization Strength
1
0.95
0.9
Accuracy
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Regularization Parameter C (log scale)


Logistic Regression: Accuracy vs Regularization Strength
1
0.9
0.8
Accuracy
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Regularization Parameter C (log scale)