CS-771 Project Report

Group: The Gradient Gang

Group Members

1. Aarsh Kaushik (Roll No: 220014)
2. Tanay Kesharwani (Roll No: 221122)
3. Nishant Kumar Meena (Roll No: 220723)
4. Siddhant Patel (Roll No: 221051)
5. Sandeep Kumar (Roll No: 220958)

1 Derivation of a Linear Model for Predicting ML-PUF Delay (8-bit)

We present a direct method to model the ML-PUF delay for an 8-bit input. The derivation leverages the differences and sums of signal delays to calculate the time for upper signal and lower signal of the PUF.

1.1 Difference of Delays

Let t_i^u and t_i^l denote the departure times of the signal from the i-th MUX pair for the upper and lower paths, respectively. Expressing t_i^u in terms of t_{i-1}^u, t_{i-1}^l, and challenge bit c_i, we have:
$t_i^u = (1 - c_i) \cdot (t_{i-1}^u + p_i) + c_i \cdot (t_{i-1}^l + s_i)(1)$
$t_i^l = (1 - c_i) \cdot (t_{i-1}^l + q_i) + c_i \cdot (t_{i-1}^u + r_i)(2)$

Subtracting (2) from (1), the delay difference at each stage is:

$$\Delta_i = t_i^u - t_i^l$$

Expanding and rearranging, we get:

$$\Delta_i = \alpha_i \cdot d_i + \beta_i$$

where

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

and

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

with p_i, q_i, r_i, s_i as system parameters.

By recursive expansion: $\Delta_0 = \alpha_0 \cdot d_0 + \beta_0$
$\Delta_1 = \alpha_1 \cdot d_1 \cdot d_0 + (\alpha_0 + \beta_1) \cdot d_0 + \beta_0 + \beta_1$
$\vdots$
$\Delta_7 = w_0x_0 + w_1x_1 + \cdots + w_7x_7 + \beta_7 = \mathbf{w}^T\mathbf{x} + b$ where

$$x_i = \prod_{j=0}^{i} d_j, \quad w_i = \alpha_i + \beta_{i-1} for i > 0, \quad w_0 = \alpha_0, \quad b = \beta_7$$

Preprint. Under review.

1.2 Sum of Delays

Adding equations (1) and (2):

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

Let $r_i' = (s_i + r_i - p_i - q_i)$, then:

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

Summing from $i = 0$ to $i = 7$:

$$t_7^u + t_7^l = t_0^u + t_0^l + \sum_{i=0}^{7}(p_i + q_i) + \sum_{i=0}^{7} c_i r_i'$$

1.3 Final Output Expression

Combining the difference and sum, the upper path delay at the output is:

$$t_7^u = \frac{1}{2}\left[(t_7^u + t_7^l) + (t_7^u - t_7^l)\right]$$

Substituting the derived forms:

$$t_7^u = \frac{1}{2}\left(\mathbf{w}^T\mathbf{x} + \beta_7 + \sum_{i=0}^{7}(p_i + q_i) + \sum_{i=0}^{7} c_i r_i'\right)$$

1.4 Feature Map Construction (8-bit, 15-Dimensional)

Let the original feature map be $\phi(\mathbf{r}) \in R^{15}$, constructed from the 8 challenge bits as:

$$\phi(\mathbf{r}) = [x_0, x_1, \ldots, x_6, 1 - 2c_0, 1 - 2c_1, \ldots, 1 - 2c_7]^T$$

where x_i are as above, and c_i are the linear terms from the sum we have transformed c_i to 1-2c_i in order to map the output to -1 and 1.Note that since x_7=$1 - 2c_7$ these two terms can absorbed in one term thus reducing the dimensionality of our feature map from 16 to 15. rest of the terms are not linearly seperable .

1.5 XOR Response Feature Map (105-Dimensional)

Let ϕ_0 and ϕ_1 be the feature maps for two different responses. The XOR operation on responses corresponds to the element-wise product of the feature maps as shown in the lectures slides that is we consider:

$$\prod_i (\mathbf{w}_i^{\mathrm{T}}\mathbf{x})$$

which is implies that our new feature map for breaking ML-PUF is where $\odot$ is the pairwise element multiplications of two vectors:

$$\Phi_{XOR} = \phi_0 \odot \phi_1$$

However, to avoid redundancy, we construct a new feature map $\Psi(\mathbf{r}_0, \mathbf{r}_1)$ of dimension 120 as follows,also note that $\phi_i(\mathbf{r}_0)$ is the value of ϕ_0 at the index i :

- 15 quadratic terms: $\phi_i(\mathbf{r}_0) \cdot \phi_i(\mathbf{r}_1)$ for $i = 1$ to 15
- 105 unique cross terms: $\phi_i(\mathbf{r}_0) \cdot \phi_j(\mathbf{r}_1)$ for $i < j$

Thus,

$$\Psi(\mathbf{r}_0, \mathbf{r}_1) = [\phi_1(\mathbf{r}_0)\phi_1(\mathbf{r}_1),\ \ldots,\ \phi_{15}(\mathbf{r}_0)\phi_{15}(\mathbf{r}_1),\ \phi_1(\mathbf{r}_0)\phi_2(\mathbf{r}_1),\ \ldots,\ \phi_{14}(\mathbf{r}_0)\phi_{15}(\mathbf{r}_1)]$$

this can be further defined as:

$$\Psi(\mathbf{c}) = \big[(1-2c_0)^2,\ (1-2c_1^2),\ \ldots,\ (1-2c_7)^2,\ x_0^2,\ \ldots,\ x_6^2,\ (1-2c_0)(1-2c_1),\ (1-2c_0)(1-2c_2),\ \ldots,\ (1-2c_0)x_6,\ (1-2c_1)(1$$

note that since x_i are functions of product of 1-2c_i terms and we also added 1-2c_i terms in our map the value of square is 1 and therefore these 15 square terms can be absorbed in bias terms hence final 105 dimensional feature map becomes

$$\Psi(\mathbf{c}) = \left[(1-2c_0)(1-2c_1),\ (1-2c_0)(1-2c_2),\ \ldots,\ (1-2c_0)x_6,\ (1-2c_1)(1-2c_2),\ \ldots,\ x_5x_6 \right]^T$$

where the cross terms are ordered lexicographically with $i < j$.

The final XOR response model is then:

$$y_{XOR} = \mathbf{W}_{XOR}^T \Psi(\mathbf{r}_0, \mathbf{r}_1) + b_{XOR}$$

where $\mathbf{W}_{XOR} \in R^{105}$ and b_{XOR} is the bias term.

2 Dimensionality of $\tilde{D}$

Based on Our Derivation in Section 1 we see that we required 15 dimensions to find the time in which upper signal or lower signal of a PUF reaches we can learn two linear models to compare the upper signals and lower signals of the ML-PUF since there is a XOR block at the end we need to obtain the pairwise multiplication of these two feature map which will have 15 square tems and 210 cross terms out which half are repeated so we are left with 105 which when added with 15 gives us a feature map 120 dimensions and since the square terms are all 1 based on how we defined a transformation from 0,1 to -1,1 we get reduction of dimensionality by 15 hence giving 105.

3 Using Kernel SVM to Break the PUF

We aim to break a machine learning PUF where the challenges are given as $c \in \{0,1\}^8$ and the response is computed as the XOR of two upper and two lower signal comparisons. In prior experiments, we showed that a linear model with an explicit feature map of dimension $d = 105$ (including cumulative products and bit transformations) is sufficient to perfectly classify the responses.

To replace the manual feature map, we propose using a kernel SVM directly on the original binary challenges.

3.1 Feature Analysis

The PUF architecture effectively relies on interactions between pairs of challenge bits:

$$response = f(c) = XOR(uppersignals, lowersignals).$$

When expanding this through the linear additive delay model, we end up with terms that depend on cumulative products of the transformed bits, mainly up to degree 2. Thus, the decision boundary lies in a space of up to second-order feature interactions.

3.2 Kernel Choice

We need a kernel that can capture at least quadratic feature interactions. We consider the following:

- **Polynomial kernel:**

  $$K(x, x') = (\gamma x^\top x' + r)^d.$$

  By setting degree $d = 2$, we directly model all linear and quadratic combinations. This perfectly matches the required interaction order.

  Recommended parameters:

  $$d = 2, \quad \gamma = \frac{1}{8}, \quad r = 1.$$

- **RBF (Gaussian) kernel:**

  $$K(x, x') = \exp(-\gamma \|x - x'\|^2).$$

The RBF kernel implicitly maps data into an infinite-dimensional feature space, making it highly expressive. It can capture the needed interactions but may be overkill for this problem.

Recommended parameter:

$$\gamma = \frac{1}{8}.$$

- **Matern kernel:** While useful in Gaussian processes, the Matern kernel is not typically used in SVMs and is not an ideal choice here.

Theoretical Justification

Since we already know that a linear SVM over a 105-dimensional feature space (including all pairwise products) perfectly classifies the data, the polynomial kernel with degree 2 is the minimal kernel choice. It ensures perfect classification while avoiding unnecessary complexity.

3.3 Final Recommendation

We recommend using: polynomial kernel with d=$\frac{1}{2}$, $\gamma = \frac{1}{8}$ and r=1. Alternatively, an RBF kernel with $\gamma = \frac{1}{8}$ can also be used.

4 Recovering Non-Negative Delays for Arbiter PUF

4.1 Model Generation as a System of 65 Linear Equations

The model generation process for a 64-bit Arbiter PUF converts 256 delays p_i, q_i, r_i, s_i for $i = 0$ to 63 into a 65-dimensional linear model $(w_0, w_1, \ldots, w_{63}, b)$. Define intermediate variables:

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

The linear model is given by:

$$w_0 = \alpha_0$$
$$w_i = \alpha_i + \beta_{i-1} \quad for \quad i = 1to63$$
$$b = \beta_{63}$$

Expressing these in terms of the delays, the system of 65 linear equations is:

1. For w_0:
$$p_0 - q_0 + r_0 - s_0 = 2w_0$$
2. For w_i, $i = 1$ to 63:
$$p_i - q_i + r_i - s_i + p_{i-1} - q_{i-1} - r_{i-1} + s_{i-1} = 2w_i$$
3. For b:
$$p_{63} - q_{63} - r_{63} + s_{63} = 2b$$

This system has 256 variables and 65 equations, making it underdetermined.

4.2 Method to Recover Non-Negative Delays

To recover non-negative delays that generate the same linear model, we use the following method:

4.2.1 Step 1: Define α and β Arrays

Define:

$$\alpha_i = w_i \quad for \quad i = 0, 1, \ldots, 63$$
$$\beta_i = \{\, 0 \; for \quad i = 0, 1, \ldots, 62 b for \quad i = 63$$

4.2.2 Step 2: Compute d_i and c_i

Compute:

$$d_i = \alpha_i + \beta_i, \quad c_i = \alpha_i - \beta_i$$

Specifically:

- For $i = 0$ to 62:
$$d_i = w_i + 0 = w_i, \quad c_i = w_i - 0 = w_i$$
- For $i = 63$:
$$d_{63} = w_{63} + b, \quad c_{63} = w_{63} - b$$

4.2.3 Step 3: Compute Delays

Set:

$$p_i = \max(d_i, 0), \quad q_i = \max(-d_i, 0)$$
$$r_i = \max(c_i, 0), \quad s_i = \max(-c_i, 0)$$

for $i = 0$ to 63.

This ensures $p_i \geq 0$, $q_i \geq 0$, $r_i \geq 0$, $s_i \geq 0$, and satisfies $p_i - q_i = d_i$, $r_i - s_i = c_i$.

4.3 Verification

To verify, compute α_i and β_i from the delays:

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

- For $i = 0$ to 62:
$$\alpha_i = \frac{w_i + w_i}{2} = w_i, \quad \beta_i = \frac{w_i - w_i}{2} = 0$$
- For $i = 63$:
$$\alpha_{63} = \frac{(w_{63} + b) + (w_{63} - b)}{2} = w_{63}, \quad \beta_{63} = \frac{(w_{63} + b) - (w_{63} - b)}{2} = b$$

Reconstruct the model:

- $w_0 = \alpha_0 = w_0$
- For $i = 1$ to 63:
$$w_i = \alpha_i + \beta_{i-1} = w_i + 0 = w_i$$
- $b = \beta_{63} = b$

The delays satisfy the original system, regenerating the same model.

5 Code to solve the ML-PUF problem by learning a linear model

The Code is Uploaded on CC sever

6 Code to solve the Arbiter PUF inversion problem to recover delays

The Code is Uploaded on the CC server

7 Performance Comparison of Linear Models with Various Hyperparameters

(a) Changing the `loss` Hyperparameter in LinearSVC

Model (Loss)	Loss Type	Total Features	Train Time (s)	Accuracy
LinearSVC	Hinge	105	0.7885	1
LinearSVC	Squared Hinge	105	1.0925	1

Inference

From the observations, it is evident that the training time for squared hinge loss is higher than that of hinge loss, despite both achieving perfect accuracy (100%). The increased training time for squared hinge loss can be attributed to its more aggressive penalty on margin violations, which requires additional computation for optimization. However, both loss functions lead to the same level of performance, demonstrating that the squared hinge loss, though more computationally expensive, does not necessarily provide a significant improvement in accuracy over hinge loss in this case and the accuracy can be solely attributed to good feature map.

(b) Changing the `C` Value (Regularization Strength)

Model	C Value	Total Features	Train Time (s)	Accuracy
LinearSVC	0.01	105	0.12	0.977
LinearSVC	10	105	1.063	1
LinearSVC	100	105	1.25	1
LogisticRegressiom	0.1	105	0.32	0.988
LogisticRegression	10	105	0.488	1
LogisticRegression	100	105	0.42	1

Inference

The experiment shows that both LinearSVC and LogisticRegression models achieve near-perfect or perfect accuracy across a wide range of C values (0.1, 10, 100), with only a minor drop in LogisticRegression at C = 0.1. The training time generally increases with higher C values, especially for LinearSVC. The consistently high accuracy indicates that the 105 mathematically derived features are highly informative and effective for breaking the ML PUF, making the models largely insensitive to the choice of regularization strength. This suggests that the feature map plays a dominant role in determining performance, while tuning C mainly affects training time rather than accuracy in this case as C increases the generalization gets stronger.

(c) Changing the `tol` (Convergence Tolerance)

Model	Tolerance Value	Total Features	Train Time (s)	Accuracy
LinearSVC	low	105	1.72	1
LinearSVC	medium	105	0.64	1
LinearSVC	high	105	0.091	1
LogisticRegression	low	105	0.676	1
LogisticRegression	medium	105	0.25	1
LogisticRegression	high	105	0.02	0.98

Inference

The results show that decreasing the tolerance value leads to an increase in training time for both LinearSVC and LogisticRegression. Despite this, the accuracy remains consistently high across all tolerance settings, except for a minor drop in LogisticRegression at high tolerance. This indicates that the mathematically derived feature map D is highly effective and provides a strong signal to the models, making them robust to convergence criteria. Therefore, even with looser stopping conditions (high tolerance), the models can achieve near-perfect accuracy, and stricter tolerance mainly affects computational cost rather than predictive performance in general accuracy increases by lowering tolerances.

(d) Changing the `penalty` Type

Model	Penalty Type	Total Features	Train Time (s)	Accuracy
LinearSVC	L-1	105	1.21	1
LinearSVC	L-2	105	1.106	1
LogisticRegression	L-1	105	1.268	1
LogisticRegression	L-2	105	0.44	1

Inference

Inference on changing L1 vs. L2 penalty:

Accuracy stays perfect (1.0) across all models and penalty types because the feature map is strong — the model can separate the classes easily regardless of the regularization type.

Training time is affected:

For both LinearSVC and LogisticRegression, L1 penalty takes more time to train than L2.

Example: LogisticRegression L1 $\rightarrow$ 1.268 s, L2 $\rightarrow$ 0.44 s.

Feature count is unchanged:

All models use all 105 features; L1 didn't shrink feature count in this setup, likely because the feature map is already well-optimized.

General explanation: L1 regularization tends to induce sparsity (drop coefficients to zero) but can be slower to converge, especially when data is already well-separated.

L2 regularization generally converges faster and provides smoother solutions but keeps all features with small weights.

Because the feature map is good here, regularization type mainly affects speed, not accuracy or feature count.