CS771: PROJECT REPORT

Team: Invictus

Ashutosh Anand **Priyanshu Gujraniya**

Akansha **Aman srivastav**

Sanskar

Abstract

This report delves into the mathematical underpinnings of Multi Level Physical Unclonable Functions (ML-PUFs) and explores their predictability using linear models. We provide a detailed derivation demonstrating that the response of a specific ML-PUF architecture can be represented by a linear model within a carefully constructed transformed feature space. Furthermore, we analyze the dimensionality of this linear model. To address potential non-linearities and enhance classification, we investigate the application of Kernel Support Vector Machines (SVMs), deriving a specific kernel function suitable for the ML-PUF problem and suggesting practical kernel implementations. Finally, we tackle the inverse problem for Arbiter PUFs, formulating a sparse linear system to recover the underlying stage delays from a given linear model and proposing a method to ensure non-negativity of the recovered delays.

1 Mathematical Proof of Linear Model Predictability for ML-PUFs

Let:

- t_i^u be the (unknown) time at which the upper signal leaves the i-th PUF
- t_i^l be the time at which the lower signal leaves the i-th PUF
- p_i, q_i, r_i, s_i, be the delay a signal takes to pass through the i-th PUF

All PUFs are different so that p_i, r_i, s_i, and q_i are distinct.

Therefore, the answer is 0 if $t_{N-1}^l < t_{N-1}^u$ and 1 otherwise. (Here, we have assumed zero-based indexing which implies $t_{-1}^u = t_{-1}^l = 0$) and N is the number of stage in the PUF

From lecture slides:

$$t_n^u = (1 - c_n)(t_{n-1}^u + p_n) + c_n(t_{n-1}^l + s_n) \tag{1}$$

$$t_n^l = (1 - c_n)(t_{n-1}^l + q_n) + c_n(t_{n-1}^u + r_n) \tag{2}$$

where $c_n \in \{0, 1\}$

Adding equations (1) and (2), we get:

$$t_n^u + t_n^l = t_{n-1}^u + t_{n-1}^l + (1 - c_n)(p_n + q_n) + c_n(r_n + s_n)$$

For $n = 0$:

$$t_0^u + t_0^l = (1 - c_0)(p_0 + q_0) + c_0(r_0 + s_0) \quad \text{(since } t_{-1}^u = t_{-1}^l = 0\text{)}$$

For $n = 1$:

$$t_1^u + t_1^l = (1 - c_0)(p_0 + q_0) + c_0(r_0 + s_0) + (1 - c_1)(p_1 + q_1) + c_1(r_1 + s_1)$$

For $n = 2$:

$t_2^u + t_2^l = (1-c_0)(p_0+q_0)+c_0(r_0+s_0)+(1-c_1)(p_1+q_1)+c_1(r_1+s_1)+(1-c_2)(p_2+q_2)+c_2(r_2+s_2)$

By induction:

$$t_7^u + t_7^l = \sum_{i=0}^{7} [(1-c_i)(p_i+q_i) + c_i(r_i+s_i)] \tag{3}$$

Let

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

from equation (1) and (2) we got

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

where:

$$\begin{aligned} d_i &= (1-2c_i) \\ \alpha_i &= \frac{(p_i - q_i + r_i - s_i)}{2} \\ \beta_i &= \frac{(p_i - q_i - r_i + s_i)}{2} \end{aligned}$$

Then,

$$\begin{aligned} \Delta_0 &= \alpha_0 d_0 + \beta_0 \\ \Delta_1 &= d_1(\alpha_0 d_0 + \beta_0) + (\alpha_1 d_1 + \beta_1) \\ \Delta_2 &= d_2 d_1(\alpha_0 d_0 + \beta_0) + d_2(\alpha_1 d_1 + \beta_1) + (\alpha_2 d_2 + \beta_2) \\ \Delta_7 &= \sum_{k=0}^{7} (\alpha_k d_k + \beta_k) \prod_{j=k+1}^{7} d_j \end{aligned} \tag{4}$$

So:

$$t_7^u - t_7^l = \sum_{i=0}^{7} [(p_i - q_i)(1-c_i) + c_i(s_i - r_i)] \prod_{j=i+1}^{7} (1-2c_j) \tag{5}$$

Adding equations (3) and (5), we get:

$$t_7^u = \frac{1}{2}\sum_{i=0}^{7} [(1-c_i)(p_i+q_i) + c_i(r_i+s_i)] + \frac{1}{2}\sum_{i=0}^{7} [(1-c_i)(p_i-q_i) + c_i(s_i-r_i)] \prod_{j=i+1}^{7} (1-2c_j) \tag{6}$$

Define:

$$\begin{aligned} d_i &= (1-2c_i) \\ A_i &= \frac{p_i + q_i + r_i + s_i}{4} \\ B_i &= \frac{p_i + q_i - r_i - s_i}{4} \\ C_i &= \frac{p_i - q_i + s_i - r_i}{4} \\ D_i &= \frac{p_i - q_i + r_i - s_i}{4} \end{aligned}$$

Simplifying equation (6):

$$t_7^u = \sum_{i=0}^{7} A_i + \gamma_7 + \sum_{i=0}^{7} d_i B_i + \sum_{i=0}^{7} (C_i + D_{i-1}) \prod_{j=i}^{7} d_j \quad (\gamma_{-1} = 0) \tag{7}$$

Let:

$$b = \sum_{i=0}^{7} A_i + C_7$$

Now, converting this to matrix form:

$$t^u(c) = W^\top \phi(c) + b$$

Absorbing the bias into $W^\top$ using the constant feature $\phi_1 = 1$:

$$t(c) = \tilde{W}^T \phi(c)$$

where:

$$\tilde{W} = (b \quad B_0 \quad \dots \quad B_6 \quad D_0 \quad (D_1 + C_0) \quad \dots \quad (D_7 + C_6 + B_7))^\top$$

$$\phi(c) = (1 \quad d_0 \quad \dots \quad d_6 \quad d_0 d_1 \dots d_7 \quad d_1 d_2 \dots d_7 \quad \dots \quad d_7)^\top$$

Differences in delay between PUF1 and PUF0 for the lower and upper paths

$$\delta_l(c) = t_1^l(c) - t_0^l(c) = (W_{1l}^T - W_{0l}^T)\phi(c) = W_{\delta^l}^\top \phi(c) \tag{8}$$

$$\delta_u(c) = t_1^u(c) - t_0^u(c) = (W_{1u}^T - W_{0u}^T)\phi(c) = W_{\delta^u}^\top \phi(c) \tag{9}$$

$$W^\top \delta^l, W^\top \delta^u \in \mathbb{R}^{17}$$

The output of the linear model in the transformed feature space is the product of the original delay differences:

$$\tilde{W}^\top \Phi(c) + \tilde{b} = -(\delta_l(c) \cdot \delta_u(c)) = -\left(W^\top \delta^l \phi(c) \cdot W^\top \delta^u \phi(c)\right)$$

$$= -\left(\sum_{i=1}^{16} (w_{\delta li}\phi_i(c) \cdot \sum_{j=1}^{16} (w_{\delta ui}\phi_j(c)\right) = \sum_{i=1}^{16}\sum_{j=1}^{16} (-(w_{\delta li}(w_{\delta ui}))(\phi_i(c)\phi_j(c)) \quad (*)$$

The construction of the initial linear models $W_{\delta^l}^\top \phi(c)$ and $W_{\delta^u}^\top \phi(c)$ already incorporates their respective bias terms. Therefore:

$$\tilde{b} = 0$$

Conclusion: The feature map $\Phi(c) = (\Phi_i(c)\Phi_j(c))1 \leq i,j \leq 16 \in \mathbb{R}^{256}$, where $\Phi_i(c)$ are components of $\phi(c)$, defines the transformed feature space. The linear model weights are W $\in \mathbb{R}^{256}$ with components $Wij = -w_{\delta li} w_{\delta uj}$. The bias is $\tilde{b} = 0$. The prediction for challenge c (assuming $c \in \{0,1\}^8$ as stated is: $r(c) = 1 + \text{sign}(W^T \Phi(c))$

2 Dimensionality of the Linear Model

From the Last section we derives that $\Phi(c) = (\Phi_i(c)\Phi_j(c))_{1 \leq i,j \leq 16} \in \mathbb{R}^{256}$. From here we can deduce that the Dimensionality of $\Phi(c)$ is 16 * 16 = 256.

3 Kernel SVM for ML-PUF Problem

3.1 Desired Feature Mapping

Given an 8-bit challenge $c = (c_1, c_2, \dots, c_8)$, the feature vector f that leads to a linear separation of the ML-PUF responses is:

$$f = (c_1, c_2, \dots, c_8, c_1c_2, c_1c_3, \dots, c_7c_8)$$

This vector has 36 dimensions, consisting of the original 8 challenge bits and the 28 pairwise products.

3.2 Kernel Definition

We investigate using a kernel SVM to classify responses for an 8-bit binary challenge $c \in \{0,1\}^8$. The goal is to find a suitable kernel.

3.3 Polynomial Feature Space

Consider a feature map $\Phi(c)$ including linear terms and unique pairwise products:

$$\Phi(c) = (c_1, \dots, c_8, c_1c_2, c_1c_3, \dots, c_7c_8)$$

This map contains 8 linear terms and $\binom{8}{2} = 28$ pairwise products (since $c_i^2 = c_i$ for binary c_i). The total dimensionality is $8 + 28 = 36$.

3.4 The Kernel Trick

A kernel function $K(c, c')$ computes the inner product $\langle \Phi(c), \Phi(c') \rangle$ in the 36-dimensional space without explicit mapping. If the data is linearly separable in this space, the kernel SVM can classify perfectly.

3.5 Kernel Derivation for $\Phi(c)$

The inner product is:

$$K(c, c') = \langle \Phi(c), \Phi(c') \rangle = \sum_{i=1}^{8} c_i c' i + \sum 1 \leq i < j \leq 8 (c_i c_j)(c_i' c_j')$$

Using $\langle c, c' \rangle^2 = \langle c, c' \rangle + 2 \sum_{1 \leq i < j \leq 8} (c_i c_i')(c_j c_j')$, we isolate the sum:

$$\sum_{1 \leq i < j \leq 8} (c_i c_j)(c_i' c_j') = \frac{1}{2}[\langle c, c' \rangle^2 - \langle c, c' \rangle]$$

Substituting back into $K(c, c')$:

$$K(c, c') = \langle c, c' \rangle + \frac{1}{2}[\langle c, c' \rangle^2 - \langle c, c' \rangle] = \frac{1}{2}\langle c, c' \rangle^2 + \frac{1}{2}\langle c, c' \rangle$$

3.6 Comparison to a Standard Polynomial Kernel

The standard polynomial kernel of degree $d = 2$ is $K_{poly}(x, y) = (\gamma \langle x, y \rangle + r)^2$.

$$K_{poly}(c, c') = \gamma^2 \langle c, c' \rangle^2 + 2\gamma r \langle c, c' \rangle + r^2$$

Matching $K(c, c')$ requires $\gamma^2 = 1/2 \implies \gamma = 1/\sqrt{2}$ and $2\gamma r = 1/2 \implies r = 1/(2\sqrt{2})$. With these parameters,

$$K_{poly}(c, c') = \frac{1}{2}\langle c, c' \rangle^2 + \frac{1}{2}\langle c, c' \rangle + \frac{1}{8}$$

This matches $K(c, c')$ up to the constant $1/8$, which does not affect the SVM decision boundary.

3.7 Conclusion and Recommendation

- **Kernel Type:** Polynomial.
- **Degree:** $d = 2$.
- **Parameters:** Equivalent to $\gamma \approx 0.707$, $r \approx 0.354$.
- **Justification:** This kernel corresponds to the inner product in the feature space $\Phi(c)$. It captures linear and pairwise interactions between challenge bits.
- **Remarks:** If data is separable in $\Phi(c)$, perfect classification is possible. Parameters should be tuned via cross-validation.

4 Linear Model for Arbiter PUF

Given a linear model with weights $w_0, w_1, \ldots, w_{k-1}$ (where $k = 64$) and a bias b, the decoding process computes intermediate values α_i and β_i, and finally the output vectors p, q, r, s.

Initialization

$$\alpha_0 = w_0$$
$$\beta_{k-1} = b$$

Iterative Calculation of α

For i from 1 to $k-1$:

$$\alpha_i = w_i - \beta_{i-1}$$

Calculation of Initial Output Vectors

For i from 0 to $k-1$:

$$\begin{aligned}
q_{hat}[i] &= 0\\
s_{hat}[i] &= 0\\
p_{hat}[i] &= \alpha_i + \begin{cases} 0 & \text{if } i = 0 \\ \beta_{i-1} & \text{if } i > 0 \end{cases} + \begin{cases} b & \text{if } i = k-1 \\ \beta_i & \text{if } i < k-1 \end{cases}\\
r_{hat}[i] &= \alpha_i - \begin{cases} 0 & \text{if } i = 0 \\ \beta_{i-1} & \text{if } i > 0 \end{cases} - \begin{cases} b & \text{if } i = k-1 \\ \beta_i & \text{if } i < k-1 \end{cases}
\end{aligned}$$

Applying Non-Negativity Constraints and Offsets

For i from 0 to $k-1$:

- If $p_{hat}[i] < 0$:

$$\begin{aligned}
offset_p &= -p_{hat}[i]\\
p_{hat}[i] &= p_{hat}[i] + offset_p = 0\\
q_{hat}[i] &= q_{hat}[i] + offset_p
\end{aligned}$$

- If $r_{hat}[i] < 0$:

$$\begin{aligned}
offset_r &= -r_{hat}[i]\\
r_{hat}[i] &= r_{hat}[i] + offset_r = 0\\
s_{hat}[i] &= s_{hat}[i] + offset_r
\end{aligned}$$

Final Output

$$\begin{aligned}
p &= p_{hat}\\
q &= q_{hat}\\
r &= r_{hat}\\
s &= s_{hat}
\end{aligned}$$

Simplified Expressions (Assuming $\beta_i = 0$ for $i < k-1$)

Under the assumption that $\beta_i = 0$ for $i = 0, 1, \ldots, k-2$, the expressions for p_{hat} and r_{hat} become:

- For $0 < i < k-1$:

$$\begin{aligned}
p_{hat}[i] &= w_i\\
r_{hat}[i] &= w_i
\end{aligned}$$

- For $i = 0$:

$$p_{hat}[0] = w_0$$
$$r_{hat}[0] = w_0$$

- For $i = k - 1$:

$$p_{hat}[k-1] = w_{k-1} + b$$
$$r_{hat}[k-1] = w_{k-1} - b$$

5 ML-PUF problem Code

Code is already Submitted

6 Arbiter PUF inversion Code

Code is already Submitted

7 Observations and Outcomes

This model was trained on `public_trn.txt` and tested on `public_tst.txt`.

7.1 Effect of chnaging loss hyperparameter in LinearSVC

Loss Function tells that how much our predicted data is away from the ground truth label. We have tested using two loss functions one is hinge loss and the other is squared hing loss.

Table 1: Results of LinearSVC

Loss Function	Average Training Time (s)	Average Test Accuracy
hinge	1.488	0.9606
squared_hinge	1.151	0.991875

7.2 Setting C in LinearSVC and LogisticRegression

Hyperparameter C: Controls the trade-off between fitting training data and preventing overfitting in models like SVM and logistic regression.

- **Small C ($C \to 0$):** Simpler model, may underfit, prioritizes generalization.
- **Large C ($C \to \infty$):** Complex model, may overfit, prioritizes fitting training data.

Optimal C is found via techniques like cross-validation to balance fitting and generalization.

Table 2: Effect of varying C on LinearSVC and LogisticRegression

Classifier	**C**	**Avg Training Time (s)**	**Avg Test Accuracy**
LinearSVC	0.01	2.0667	0.99625
	10	3.17845	1
	100.0	2.259	0.9943
LogisticRegression	0.001	0.5824	0.80625
	1.0	0.7382	0.99625
	100.0	0.7069	1

7.3 Changing tol in LinearSVC and LogisticRegression

Tolerance Parameter (tol): Sets the stopping criterion for iterative machine learning algorithms based on the change in the model or its performance between iterations.

- **Small tol:** Higher precision, longer training time.
- **Large tol:** Faster training time, potentially lower precision.

tol manages the balance between accuracy and training duration by defining convergence.

Table 3: Effect of varying tolerance parameter on LinearSVC and LogisticRegression

Classifier	Tolerence	Avg Training Time (s)	Avg Test Accuracy
LinearSVC	1×10^{-2}	1.517	0.99625
	1×10^{-3}	1.816	0.99625
	1×10^{-3}	2.5814	0.99625
LogisticRegression	1×10^{-2}	0.7588	1
	1×10^{-3}	0.8045	1
	1×10^{-4}	0.8513	1

7.4 Effect of changing the penality(regularization) hyperparameter

L1 and L2 Regularization: Techniques to prevent overfitting by penalizing model complexity.

- **L2 (Ridge):**
  - Penalty: $\propto ||\mathbf{w}||_2^2 = \sum w_i^2$ (squared magnitude of weights).
  - Effect: Shrinks weights towards zero.
  - Feature Selection: No automatic selection.
- **L1 (Lasso):**
  - Penalty: $\propto ||\mathbf{w}||_1 = \sum |w_i|$ (absolute magnitude of weights).
  - Effect: Drives some weights to exactly zero (sparse models).
  - Feature Selection: Automatic feature selection.

The **regularization strength** controls the penalty. L1 for feature selection, L2 for weight shrinkage.

Table 4: Effect of varying penalty type on LinearSVC and LogisticRegression

Classifier	Penalty	Avg Training Time (s)	Avg Test Accuracy
LinearSVC	l1	4.751	1
	l2	2.526	0.99625
LogisticRegression	l1	2.06399	1
	l2	0.6449	1