Companion Arbiter PUFs

Abhishek Bharti (210035) babhishek21@iitk.ac.in
Medha Srivastava (210603) medhas21@iitk.ac.in
Mukesh Nath (210639) mukeshn21@iitk.ac.in
Rishi Rakesh Agrawal (210852) rishira21@iitk.ac.in
Rishit Bhutra (210857) rishitb21@iitk.ac.in
Naman Jain (200618) jnaman20@iitk.ac.in

Group No. 35: Pixel Prophets

Abstract

This report examines the security of Companion Arbiter PUFs (CAR-PUFs) against linear machine-learning attacks. CAR-PUFs, proposed by Melbo, compare timing differences between signals from multiple arbiter PUFs against a secret threshold value to generate responses. We demonstrate the existence of a linear model capable of accurately predicting CAR-PUF responses and develop a methodology for learning this model using provided challenge-response pairs. Experimental results with LinearSVC and LogisticRegression classifiers highlight the impact of hyperparameters on training time and accuracy, shedding light on CAR-PUFs' vulnerability to linear machine learning attacks.

1 Mathematical Formulation

We have a car PUF that uses 2 arbiter PUFs – a working PUF and a reference PUF, as well as a secret threshold value $\tau > 0$. Let Δ_w, Δ_r be the difference in timings experienced for the two PUFs on the same challenge. The response to this challenge is 0 if $|\Delta_w - \Delta_r| \leq \tau$ and the response is 1 if $|\Delta_w - \Delta_r| > \tau$ where $\tau > 0$ is the secret threshold value.

We start with the expression for Δ_{31}:

$$\Delta_{31} = w_0 \cdot x_0 + w_1 \cdot x_1 + \ldots + w_{31} \cdot x_{31} + \beta_{31} = \mathbf{W}^T\mathbf{X} + b$$

where

$$x_i = d_i \cdot d_{i+1} \cdot \ldots \cdot d_{31}$$
$$w_0 = \alpha_0$$
$$w_i = \alpha_i + \beta_{i-1} \quad (\text{for } i > 0)$$

with

$$d_i = (1 - 2c_i) \quad \text{where } c_i \text{ is the } i\text{-th index of the challenge vector } \mathbf{C}$$

and

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

If $\Delta_{31} < 0$, the upper signal wins and the answer is 0.
If $\Delta_{31} > 0$, the lower signal wins and the answer is 1.
Therefore, the answer is given by:

$$\text{Answer} = \frac{1 + \text{sign}(\mathbf{W}^T\mathbf{X} + b)}{2}$$

Now we will implement the same model for our both working and reference PUFs:
Let $(\mathbf{u}, p), (\mathbf{v}, q)$ be the two linear models that can exactly predict the outputs of the two arbiter

PUFs sitting inside the CAR-PUF.

For the working PUF, we have:

$$\Delta_w = w_0 \cdot x_0 + w_1 \cdot x_1 + \ldots + w_{31} \cdot x_{31} + \beta_{31} = \mathbf{u}^T \mathbf{x} + p$$

And for the reference PUF:

$$\Delta_r = w_0 \cdot x_0 + w_1 \cdot x_1 + \ldots + w_{31} \cdot x_{31} + \beta_{31} = \mathbf{v}^T \mathbf{x} + q$$

If $|\Delta_w - \Delta_r| - \tau \leq 0$, the answer is 0. If $|\Delta_w - \Delta_r| - \tau > 0$, the answer is 1. Since the same challenge is used for both PUFs, we have:

$$|\Delta_w - \Delta_r| - \tau = \left|(\mathbf{u} - \mathbf{v})^T \mathbf{x} + p - q\right| - \tau$$

The response is given by:

$$\text{Response} = \frac{1 + \text{sign}(|\Delta_w - \Delta_r| - \tau)}{2} = \frac{1 + \text{sign}(\left|(\mathbf{u} - \mathbf{v})^T \mathbf{x} + p - q\right| - \tau)}{2}$$

Now, let $\Delta = \Delta_w - \Delta_r = (\mathbf{u} - \mathbf{v})^T \mathbf{x} + p - q$. If we multiply $|\Delta| - \tau$ by a positive number, the sign of $|\Delta_w - \Delta_r| - \tau$ remains the same. Let's multiply it by $|\Delta| + \tau$, which is always positive:

$$(|\Delta| - \tau)(|\Delta| + \tau) = \Delta^2 - \tau^2$$

So, we can write the response as:

$$\text{Response} = \frac{1 + \text{sign}(|\Delta_w - \Delta_r| - \tau)}{2} = \frac{1 + \text{sign}(\Delta^2 - \tau^2)}{2}$$

Let $\mathbf{w} = (\mathbf{u} - \mathbf{v})^T$ and $k = p - q$. Then,

$$\Delta^2 = (\mathbf{w}^T \mathbf{x} + k) \cdot (\mathbf{w}^T \mathbf{x} + k)$$

Expanding this expression, we get:

$$\Delta^2 = \sum_{i=0}^{31} w_i \cdot x_i^2 + \sum_{i=0}^{31} 2kw_i \cdot x_i + \sum_{i \neq j}^{31} \sum_{j=0}^{31} 2w_i w_j \cdot x_i \cdot x_j + k^2$$

This equation has a total of $32 + 32 + 496 + 1 = 561$ terms, but ${x_i}^2$ terms will merge into a constant since $x_i = \pm 1 \Rightarrow$ its square will always be 1. Let this whole constant be denoted as K, which further will merge into the τ^2 term and make a new constant. In such a way, $\Delta^2 - \tau^2$ has $561 - 32 - 1 = 528$ variables in the form of the 2nd and 3rd terms in the above equation. Thus,

$$\Delta^2 - \tau^2 = \sum_{i=0}^{31} 2kw_i \cdot x_i + \sum_{i \neq j}^{31} \sum_{j=0}^{31} 2w_i w_j \cdot x_i \cdot x_j + b$$

where $b = K - \tau^2$.

Substituting this into the response expression, we get:

$$\text{Response} = \frac{1 + \text{sign}(\Delta^2 - \tau^2)}{2}$$

Comparing with the given response format

$$\text{Response} = \frac{1 + \text{sign}(\mathbf{W}^\top \phi(\mathbf{c}) + b)}{2}$$

Finally, We get $\phi(c) = (x_0, x_1, \ldots, x_{31}, x_0 x_1, \ldots, x_{30} x_{31})$. The dimensionality of the function $\phi(c)$ is 528.

2 Experimental Outcomes:

2.1 (a) Changing the loss hyperparameter in LinearSVC (hinge vs squared hinge)

Table 1: Effect of Loss Hyperparameter

Loss Hyperparameter	Training Time	Test Misclassification Rate
Hinge Loss	19.44174492	0.0103
Square Hinge Loss	34.579272	0.00812

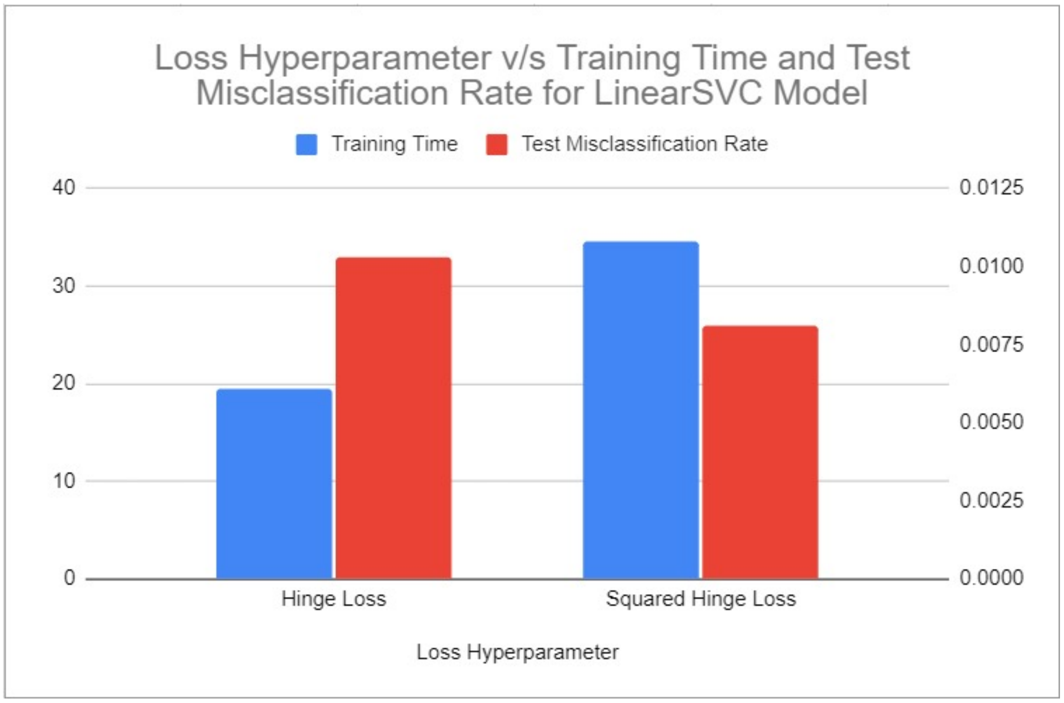

Figure 1: Effect of Loss Hyperparameter

Observation: In general, training with squared hinge loss is expected to take longer than training with hinge loss. This is because squared hinge loss involves computing squared terms, which are generally more computationally intensive compared to hinge loss, which involves only linear terms. The squared hinge loss function penalizes outliers more heavily compared to hinge loss, which can lead to a more complex optimization problem. Consequently, the optimization process may require more iterations to converge when using squared hinge loss, leading to longer training times.

2.2 (b) Setting C in LinearSVC and LogisticRegression to high/low/medium values

Table 2: Effect of C Values on LinearSVC Model

C Values	Training Time	Test Misclassification Rate
0.01	4.55374648	0.0135
0.1	13.22803983	0.0101
1	18.9645034	0.00816
10	18.07491058	0.00792
50	18.00416077	0.00846
100	18.337205	0.00826
500	17.92924791	0.00864
1000	18.17217189	0.0092

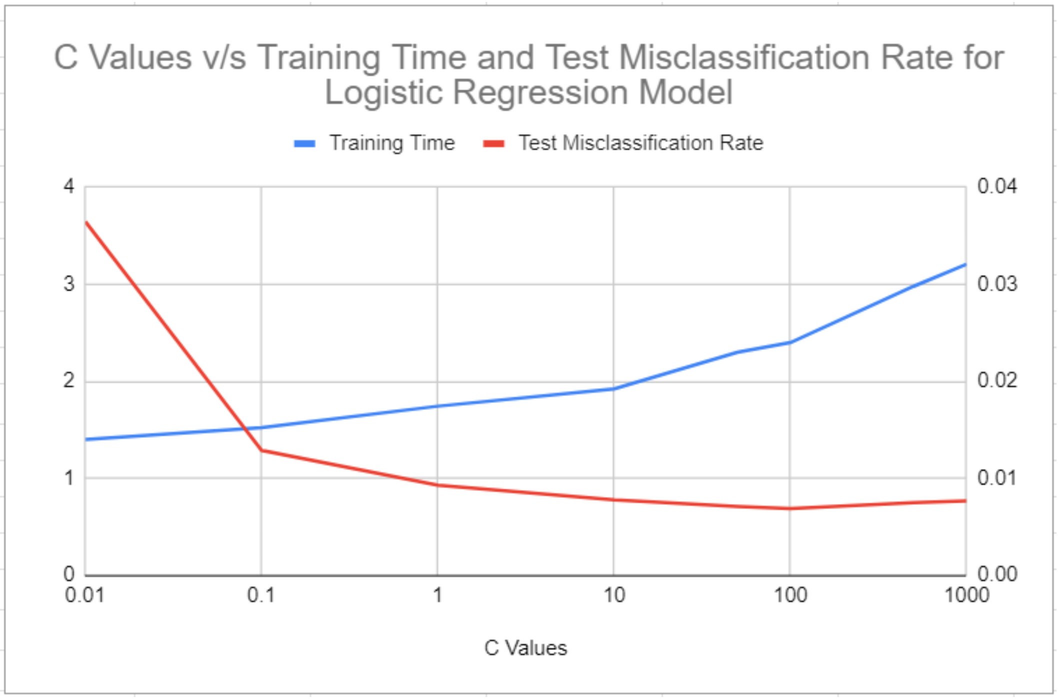

Figure 2: LinearSVC

Observation: The C parameter controls the penalty for misclassification, with smaller values of C leading to stronger regularization and potentially simpler models, while larger values of C allow the model to fit the training data more closely, potentially leading to longer training times.
The training time exhibit an increasing trend as C values increase due to the increased complexity and potential overfitting of the model.

Table 3: Effect of C Values on Logistic Regression Model

C Values	Training Time	Test Misclassification Rate
0.01	1.40262406	0.0365
0.1	1.52405432	0.0129
1	1.745059878	0.0093
10	1.922780091	0.0078
50	2.300209421	0.0071
100	2.400838354	0.0069
500	2.981411726	0.0075
1000	3.209210138	0.0077

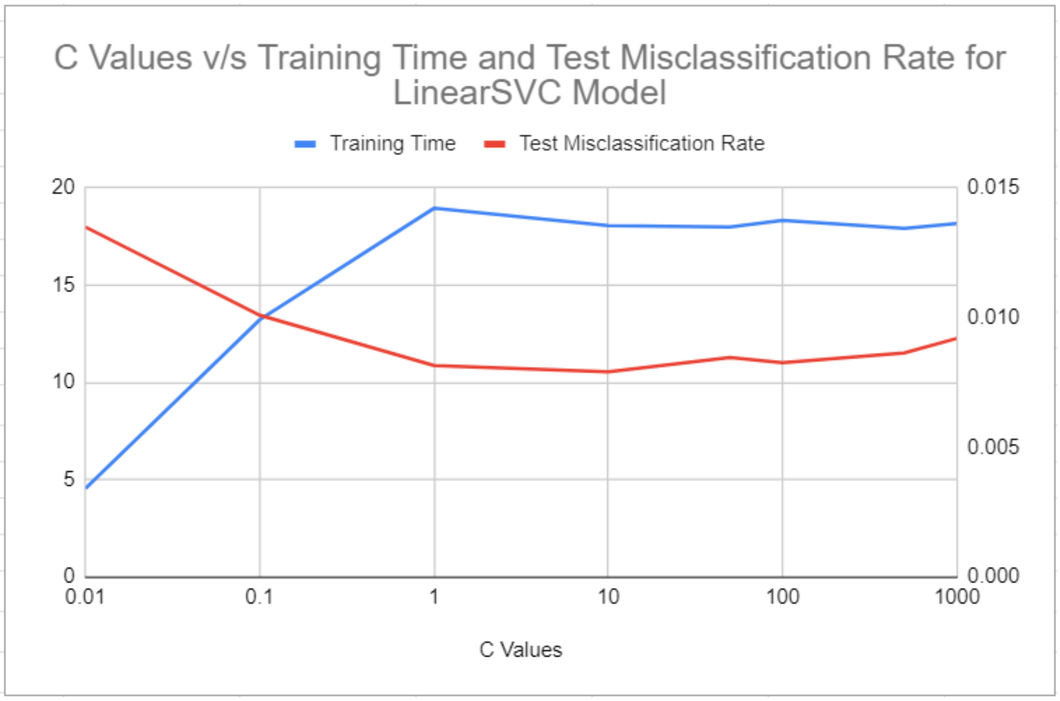

Figure 3: Logistic Regression

Observation: Overall, training time may exhibit a somewhat similar trend to LinearSVC models, where training time increases with higher values of C due to the increased complexity and potential overfitting of the model.