# CS771

The Neural Nuts

Febraury 2023

## Problem 1

In this question, we will give a detailed mathematical derivation for how a simple XORRO PUF can be broken by a single linear model. Recall that a simple XORRO PUF has just two XORROs and has no select bits and no multiplexers. The challenge to a simple XORRO PUF has just R bits.
We will first define a map $\phi$: $\{0,1\}^R \rightarrow R^D$ mapping R-bit 0/1-valued challenge vectors to D-dimensional feature vectors (for some $D > 0$). We will then show that for any simple XORRO PUF, there exists a linear model (i.e., $w \in R^D, b \in R$) such that for all challenges $c \in \{0,1\}^R$, the following expression gives the correct response:

$$\frac{1 + sign(\mathbf{w^T}\phi(\mathbf{c}) + b)}{2}$$

**Solution :** Given a set of N responses generated by an XORRO PUF, it is possible to break the system by using a single linear model. This is because the XOR operation can be written as a linear function of the input parameters.Now, if frequency of 1st XORRO is greater than frequency of 2nd XORRO, then time period of 2nd wil be greater and hence their difference will have opposite sign as of $\Delta T$ . So, we can use time period difference as well. The mathematical derivation of the single linear model to break an XORRO PUF can be done as follows:

Let, $b_i$ = the upper signal entering the i-th XOR
$a_i$ = the lower signal entering the i-th XOR
$b_{i+1}$ = the output signal of the i-th XOR
Let $\delta^i_{a_i b_i}$ be time taken by the signal to pass through he i-th XOR Gate for input $(a_i, b_i)$ Now, after each half cycle as the input gets flipped, hence the upper signal $b_i$ will be both 0 and 1 for half time periods
Let $t_0$ correspond to case when $b_i$ will be 1 and $t_1$ to when $b_i$ will be 1.

$$t_0 + t_1 = (\delta^i_{0a_i}) + (\delta^i_{1a_i})$$

Let $\mathrm{T}_i = t_0 + t_1$
Therefore,
$\mathrm{T}_0 = \delta^0_{0a_i} + \delta^0_{1a_i}$
$\mathrm{T}_1 = \delta^1_{0a_i} + \delta^1_{1a_i}$
Similarly, we can find $\mathrm{T}_3, T_4, ...$
Hence, total time T is
$\mathrm{T} = \sum_{i=0}^{R-1} \left( (\delta^i_{0a_i}) + (\delta^i_{1a_i}) \right)$
Now,
$\Delta T = \sum_{i=0}^{R-1} \left( (\delta^i_{0a_i}) - (\delta'^i_{0a_i}) \right) + \sum_{i=0}^{R-1} \left( (\delta^i_{1a_i}) - (\delta'^i_{1a_i}) \right)$ (1)

Where $\Delta$T denotes the lag difference and $\delta'$ is the time delay due to the other XORRO Let,
$d^i_0 = \delta^i_{00} + \delta^i_{10}$ , where $a_i = 0$
$d^i_1 = \delta^i_{01} + \delta^i_{11}$ , where $a_i = 1$

$T$ in terms of $d^i$ can be defined as:

$$T = \sum_{i=0}^{R-1} (d^i_{a_i})$$

Now we can say that, $d^i_{a_i} = a_i(d^i_1 - d^i_0) + d^i_0$

$$T = \sum_{i=0}^{R-1} (a_i(d^i_1 - d^i_0) + d^i_0)$$

Therefore,

$$\Delta T = \sum_{i=0}^{R-1} ((1 - a_i)(d^i_0) + a_i(d^i_1)) - ((1 - a_i)(d'^i_0) + a_i(d'^i_1))$$

where $d'$ is for the second XORRO,

$$\Delta T = \sum_{i=0}^{R-1} (a_i((\delta^i_{01}+\delta^i_{11}-\delta^i_{00}-\delta^i_{10})-(\delta'^i_{01}+\delta'^i_{11}-\delta'^i_{00}-\delta'^i_{10}))+(\delta^i_{00}+\delta^i_{10}-\delta'^i_{00}-\delta'^i_{10}))$$

Hence, we can see that it is a linear function of $a_i$ with a bias term b

Writing the weights associated with $a_i$ and bias in terms of above defined variables :

-w$_i = (\delta^i_{01} + \delta^i_{11} - \delta^i_{00} - \delta^i_{10}) - (\delta'^i_{01} + \delta'^i_{11} - \delta'^i_{00} - \delta'^i_{10})$
-b $= \sum_{i=0}^{R-1} [\delta^i_{00} + \delta^i_{10} - \delta'^i_{00} - \delta'^i_{10}]$
Finally, we get

$$(-\Delta T) = w^T(c) + b$$

where $(c)$ is the challenge vector
Therefore,

$$\phi(c) = c$$

Therefore, $\phi$ is linear function and it is evident from above equations, if r=1,then $(\Delta T) < 0$ and if r=0, then $(\Delta T) > 0$. Hence proved.

## 2. Extending above linear model to crack an Advanced XORRO PUF

**Solution :**
In advanced XORRO PUF, for each data-point $x_i$, we have a challenge pair $a_i = [x_{i1}, x_{i2}, ..., x_{i64}], p_i = [x_{i65}, x_{i66}, ..., x_{i68}]$ and $q_i = [x_{i69}, .., x_{i72}]$

For $x_i$, we will create a feature vector $Z_i \in R^{1040}$. It basically represents 16 groups of 65 features (16*65=1040), where the $k^{th}$ group $\in R^{16}$ represents the parameters of $k^{th}$ XOR gate. The 65 features represents 1 bias term and 64 elements of challenge vector. Now, let m be the decimal equivalent of p and n be of q.

We will set

$$[Z_{i65m}, .., Z_{i65(m+1}] = [1, a_1, .., a_{64}]$$

and

$$[Z_{i65n}, .., Z_{i65(n+1}] = -[1, a_1, ..., a_64]$$

and, other

$$z_{ij} = 0$$

Now, let the parameter vector be $w \in R^{1040}$
Now,

$$wz_i^T = \sum_{k=65m}^{65(m+1)} w_k z_{ik} + \sum_{k=65n}^{65(n+1)} w_k z_{ik}$$

(from Q1)

$$= [\sum_{k=65m+1}^{65(m+1} w_k x_{ik} + w_{65m}] - [\sum_{k=65n+1}^{65(n+1} w_k x_{ik} + w_{65n}]$$

$$= T_{a_i,p_i} - T_{a_i,q_i}$$
$$= \Delta T$$

(where $T_{a_i,pi}$ denotes the delay caused due to input challenge $a_i$ for $p^{ith}$ XORRO)

Hence, clearly it is a linear function of $z_i$, and the sign of $\Delta T$ can be used to predict output as 1 or 0 (similar to Q1).

Therefore, we define a function $\varphi(a_i, pi, qi)$ which transforms it to $z_i$ using definition defined above and we get a linear function of $z_i$.
We can now use any Linear Machine Learning model to determine the parameters $w_i$ and hence break the advance XORRO PUF.

# 3. Implemented in submit.py

# 4. Experiments and Observations:-

4a) On changing the loss hyperparameter in LinearSVC (hinge vs squared hinge)

Table 1: On changing the loss function

| **Quantity** | **Hinge** | **Sqaured Hinged** |
|---|---|---|
| **Accuracy** | 0.98977 | 0.99192 |
| **Training Time** | 4.810038 | 4.598184 |
| **Testing Time** | 1.19855 | 1.13152 |
| **Disk Size** | 8913.0 | 8921.0 |

Hence, we can see that Squared hinge has performed better than Hinge loss

4b) On varying the value of the parameter C:

Table 2: For LinearSVC (hinge loss)

| **C val** | **Training Time(s)** | **Test Accuracy** |
|---|---|---|
| 0.01 | 3.140 | 0.9595 |
| 10 | 4.810 | 0.9897 |
| 100 | 4.413 | 0.9896 |
| 10000 | 4.864 | 0.9894 |

Table 3: For Logistic Model

| **C val** | **Training Time(s)** | **Test Accuracy** |
|---|---|---|
| 0.01 | 7.368 | 0.9351 |
| 10 | 11.009 | 0.9915 |
| 100 | 22.863 | 0.9927 |
| 10000 | 87.806 | 0.9925 |

Hence, we can see that performance of both the models is peaking around **C**=100

4c) On varying the value of the parameter tol:

Table 4: For LinearSVC (hinge loss)

| **tol val** | **Training Time(s)** | **Test Accuracy** |
|---|---|---|
| 1e-6 | 5.148 | 0.98987 |
| 1e-4 | 5.667 | 0.98997 |
| 1e-2 | 4.413 | 0.99022 |
| 1 | 4.864 | 0.98955 |

Table 5: For Logistic Model

| **tol val** | **Training Time(s)** | **Test Accuracy** |
|---|---|---|
| 1e-6 | 23.198 | 0.99145 |
| 1e-4 | 26.305 | 0.99145 |
| 1e-2 | 22.338 | 0.99145 |
| 1 | 10.942 | 0.99147 |

For **LinearSVC**, the optimum value of tolerance for both training time and test accuracy is around 0.01

While for **Logistic**, the optimum value of tolerance for both training time and test accuracy is around 1.

4d) On chaning the penalizing term

Table 6:

| **Regularization** | **Training Time(s)** | **Test Accuracy** |
|---|---|---|
| l1 (SVC) | 34.063 | 0.99135 |
| l2 (SVC) | 4.094 | 0.990725 |
| l1 (Logistic) | 57.383 | 0.992525 |
| l2 (Logistic) | 24.918 | 0.99145 |

L1 results in a little-bit better accuracy but it comes at a substantially train time cost, in comparsion to L2 for both the models.