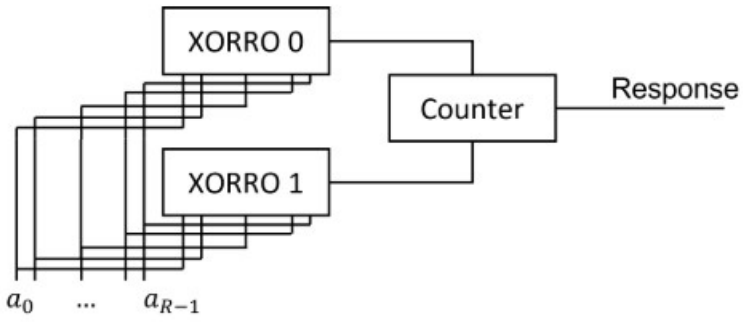
CS771 Assignment 1
Team: Spectre

Group Members:

Raghav Karan - 200749
Subhrajit Mishra - 201006
Suryanshu Kumar Jaiswal - 201025
Harsh Chaudhary - 200410
Aastik Guru - 200007

1 Simple XORRO PUF

Given that
a_0,a_1,a_2,.....,a_{R-1} are the set configuration bit for any particular XORRO PUF, with a_i being the input to the i-th xor gate of the PUF.



As we know, PUF will give different delay based on input and configuration bit supplied to it.

Lets define the delay in any particular XOR gate of XORRO PUF
δ_{00},δ_{01},δ_{10},δ_{11} are the delay based on input and configuration bit.

S. No.	Input bit	Configuration bit	Output
1	0	0	δ_{00}
2	0	1	δ_{01}
3	1	0	δ_{10}
4	1	1	δ_{11}

$\delta^0_{0a_0}$ Implies delay in XORRO(0) for input 0 and configuration bit a_0
$\delta^0_{1a_0}$ Implies delay in XORRO(0) for input 1 and configuration bit a_0

$\delta^0_{0a_1}$ Implies delay in XORRO(0) for input 0 and configuration bit a_1
$\delta^0_{1a_1}$ Implies delay in XORRO(0) for input 1 and configuration bit a_1
.
.
.
.
$\delta^0_{0a_{R-1}}$ Implies delay in XORRO(0) for input 0 and configuration bit a_{R-1}
$\delta^0_{1a_{R-1}}$ Implies delay in XORRO(0) for input 1 and configuration bit a_{R-1}

Similarly

Preprint. Under review.

$\delta^1_{0a_0}$ Implies delay in XORRO(1) for input 0 and configuration bit a_0
$\delta^1_{1a_0}$ Implies delay in XORRO(1) for input 1 and configuration bit a_0

$\delta^1_{0a_1}$ Implies delay in XORRO(1) for input 0 and configuration bit a_1
$\delta^1_{1a_1}$ Implies delay in XORRO(1) for input 1 and configuration bit a_1
.
.
.
.
$\delta^1_{0a_{R-1}}$ Implies delay in XORRO(1) for input 0 and configuration bit a_{R-1}
$\delta^1_{1a_{R-1}}$ Implies delay in XORRO(1) for input 1 and configuration bit a_{R-1}

Total time period

For XORRO(0):

$$(\delta^0_{0a_0}+\delta^0_{1a_0})+(\delta^0_{0a_1}+\delta^0_{1a_1})+ .... + (\delta^0_{0a_{R-1}}+\delta^0_{1a_{R-1}})$$

For XORRO(1):

$$(\delta^1_{0a_0}+\delta^1_{1a_0})+(\delta^1_{0a_1}+\delta^1_{1a_1})+ .... + (\delta^1_{0a_{R-1}}+\delta^1_{1a_{R-1}})$$

The set of configuration bits must have odd number of 1's for the XORRO PUF to act as an inverter. As a result, if the input to any XOR gate is 1 in the first half cycle then it must be 0 for the next half cycle. So, to calculate the total delay(thus the frequency), we can safely add δ_{0a_i} and δ_{1a_i} for $i \in [0, R-1]$
Counter will output 0 if XORRO(1) is faster and output 1 if XORRO(0) is faster. Hence analysing difference in delay of XORRO(1) from XORRO(0):

$$\text{f}(a_0)= (\delta^1_{0a_0}+\delta^1_{1a_0}) - (\delta^0_{0a_0}+\delta^0_{1a_0})$$
$$\text{f}(a_1)= (\delta^1_{0a_1}+\delta^1_{1a_1}) - (\delta^0_{0a_1}+\delta^0_{1a_1})$$
$$\vdots$$
$$\text{f}(a_i)= (\delta^1_{0a_i}+\delta^1_{1a_i}) - (\delta^0_{0a_i}+\delta^0_{1a_i})$$
$$\vdots$$
$$\text{f}(a_{R-1})= (\delta^1_{0a_{R-1}}+\delta^1_{1a_{R-1}}) - (\delta^0_{0a_{R-1}}+\delta^0_{1a_{R-1}})$$

Total difference in delay

$$\sum_{i=0}^{R-1} \text{f}(a_i)$$
here R=64, which means
$$\sum_{i=0}^{63} \text{f}(a_i)$$

Therefore

$$\Delta\text{T}=\sum_{i=0}^{63}(1-a_i)\ \text{f}(a_i=0)+ \sum_{i=0}^{63}(a_i)\ \text{f}(a_i=1)$$
$$\Delta\text{T}=\sum_{i=0}^{63}\text{f}(a_i=0)+ \sum_{i=0}^{63} a_i\ [\text{f}(a_i=1) - \text{f}(a_i=0)]$$

By looking above equation, we can say that $\sum_{i=0}^{63}\text{f}(a_i=0)$ is constant which can be defined as some constant $b \in R$

$$\text{b=f}(a_0=0)+\text{f}(a_1=0)+\text{f}(a_2=0)+\text{f}(a_3=0)+.....+\text{f}(a_{63}=0)$$

Now ΔT can be written as,

$$\Delta\text{T= b} + \sum_{i=0}^{63} a_i\ [\text{f}(a_i=1) - \text{f}(a_i=0)]$$

$$\Delta T = b + [a_0\ a_1\ a_2\ .....\ a_{63}].\begin{bmatrix} [f(a_0=1) - f(a_0=0)] \\ [f(a_1=1) - f(a_1=0)] \\ [f(a_2=1) - f(a_2=0)] \\ \vdots \\ [f(a_{63}=1) - f(a_{63}=0)] \end{bmatrix}$$

Here $[a_0\ a_1\ a_2\ .....\ a_{63}]$ is the provided variable configuration bit.
Define

$$W^T = \begin{bmatrix} [f(a_0=1) - f(a_0=0)] \\ [f(a_1=1) - f(a_1=0)] \\ [f(a_2=1) - f(a_2=0)] \\ \vdots \\ [f(a_{63}=1) - f(a_{63}=0)] \end{bmatrix}$$

and

$$\phi(c) = [a_0\ a_1\ a_2\ .....\ a_{63}]$$

Final equation can be written as,

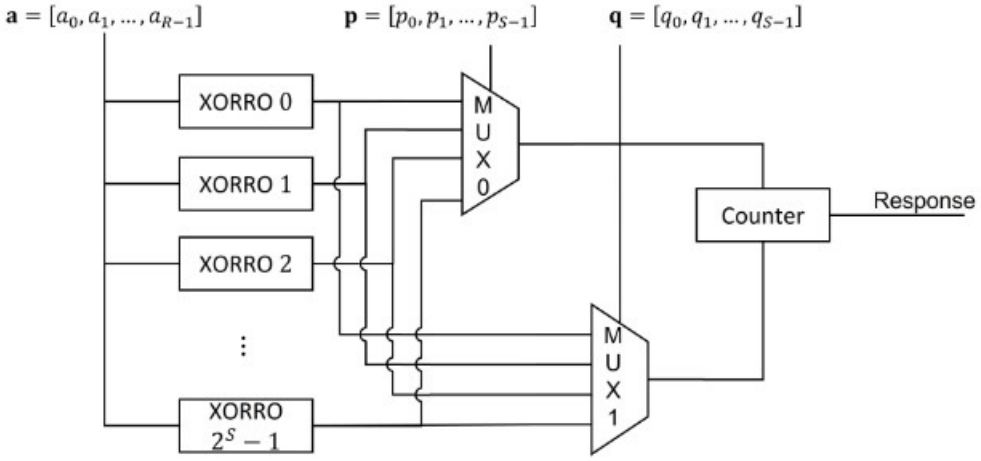
$$\Delta T = b + W^T.\ \phi(c)$$

If $\Delta T > 0$ then response is 1 and If $\Delta T < 0$ then response is 0. Hence response of the system can be expressed as:

$$\textbf{Response} = \frac{1 + \text{sign}(b + W^T.\ \phi(c))}{2}$$

This is our linear model to crack a simple XORRO PUF.

2 Advanced XORRO PUF



This question is an extension of the previous question. Here we have 16 XORRO PUFs out of which, we select any two based on the select bits given to the MUXs. We initialise a dataset that contains unique models for each pair of PUFs. Another dataframe containing the datasets specific to each model is stored.

We run a "for-loop" that interprets the responses of the MUXs and organises the data into specific datasets for specific models. Another loop extracts and trains the model with its specific dataset. Note that the model for two XORRO PUFs is linear and is explained in the previous question.

The total number of XORRO pairs possible from two 'n' bit MUXs is:

$$^{2^n}P_2 = (16 * 15) = 240 \qquad , n = 4\ \ in\ our\ case.$$

An initial analysis leads to a natural conclusion that there is a need of 240 model for all the possible permutations of XORRO PUFs. But further inspection suggets that the cases having inputs from MUX1 and MUX2 interchanged, can be predicted using a single model.

For example: Consider the outputs from (MUX1=4 ,MUX2=7) and (MUX1=7 ,MUX2=4). It is evident that the outputs for both the pairs will be complementary to each other. Thus, rather than training two different models, a single model can be trained, with a small modification of output inversion. Therfore, we require

$$^{16}C_2 = ((16 * 15)/2) = 120$$

3 Creating the Model

my_fit(): This function takes in the training data as the argument and outputs a set of models trained for each pair of XORRO PUFs. The model is prepared in python and uses numpy and sklearn libraries to execute its function.

my_predict(): This function takes in the test data, and the model trained from the previous my_fit() and outputs the predicted set of responses for each input in the test set.

4 Comparisions

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

S. No.	Loss Hyperparameter	Accuracy(%)
1	hinge	94.1125
2	squared hinge	94.5175

Part(b): Setting c hyperparameter in LinearSVC and LogisticRegression to high/low/medium values

LinearSVC

S. No.	c Hyperparameter	Accuracy(%)
1	1(low)	94.7425
2	20(medium)	94.2
3	40(high)	94.0275

Logistic Regression

S. No.	c Hyperparameter	Accuracy(%)
1	1(low)	93.9175
2	20(medium)	94.9825
3	40(high)	94.985

Part(c): changing the tol hyperparameter in LinearSVC and LogisticRegression to high/low/medium values

LinearSVC

S. No.	tol Hyperparameter	Accuracy(%)
1	10^{-10} (low)	94.4475
2	10^{-4} (medium)	94.5175
3	10^{1} (high)	87.9875

Logistic Regression

S. No.	tol Hyperparameter	Accuracy(%)
1	10^{-9} (low)	94.9975
2	10^{-2} (medium)	95.005
3	10^{1} (high)	91.1375

References

1. https://scikit-learn.org/stable/
2. https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
3. Documentary scikit-learn logistic Regression