OpenBox CS771

May 5, 2025

1 MATHEMATICAL DERIVATION

Starting with the upper signal equations for two PUF instances (denoted as PUFO and PUF1). For multiplexer index $i = 1$, the upper signal equations are:

PUF0 ($n = 0$):

$$ {}^0t_1^u = (1 - c_1)\left[{}^0t_0^u + {}^0p_1\right] + c_1\left[{}^0t_0^l + {}^0s_1\right] $$

PUF1 ($n = 1$):

$$ {}^1t_1^u = (1 - c_1)\left[{}^1t_0^u + {}^1p_1\right] + c_1\left[{}^1t_0^l + {}^1s_1\right] $$

We then do the subtraction of Upper Signals at MUX1 by defining the difference in upper signals at MUX1 as:

$$ \Delta t_1^u = {}^1t_1^u - {}^0t_1^u $$

Substitute the equations for ${}^1t_1^u$ and ${}^0t_1^u$:

$$ \begin{aligned} \Delta t_1^u &= \left[(1 - c_1)({}^1t_0^u + {}^1p_1) + c_1({}^1t_0^l + {}^1s_1)\right] \\ &\quad - \left[(1 - c_1)({}^0t_0^u + {}^0p_1) + c_1({}^0t_0^l + {}^0s_1)\right] \\ &= (1 - c_1)\underbrace{({}^1t_0^u - {}^0t_0^u)}_{\Delta t_0^u} + c_1\underbrace{({}^1t_0^l - {}^0t_0^l)}_{\Delta t_0^l} \\ &\quad + (1 - c_1)\underbrace{({}^1p_1 - {}^0p_1)}_{\epsilon_1^u} + c_1\underbrace{({}^1s_1 - {}^0s_1)}_{\lambda_1^u + \epsilon_1^u} \end{aligned} $$

where:

- $\epsilon_1^u = {}^1p_1 - {}^0p_1$ (Intrinsic parameter mismatch)
- $\lambda_1^u = {}^1s_1 - {}^0s_1 - \epsilon_1^u$ (Net parameter mismatch)

After rearranging the terms we get the upper signal time difference at MUX1 as:

$$ \Delta t_1^u = \Delta t_0^u + c_1\left(-\Delta t_0^u + \Delta t_0^l\right) + \epsilon_1^u + c_1\lambda_1^u $$

We then define the Delta Difference at MUX0 as:

$$ {}^1\Delta_0 - {}^0\Delta_0 = \underbrace{({}^1t_0^u - {}^0t_0^u)}_{\Delta t_0^u} - \underbrace{({}^1t_0^l - {}^0t_0^l)}_{\Delta t_0^l} $$

where:

- ${}^1\Delta_0 = {}^1t_0^u - {}^1t_0^l$ (Upper-lower difference for PUF1 at MUX0)

- ${}^{0}\Delta_0 = {}^{0}t_0^u - {}^{0}t_0^l$ (Upper-lower difference for PUF0 at MUX0)

The difference between upper-lower signal mismatches for PUF1 and PUF0 at MUX0:

$$\boxed{{}^{1}\Delta_0 - {}^{0}\Delta_0 = \Delta t_0^u - \Delta t_0^l}$$

Now after elimination of Δt_0^l, We solve for Δt_0^l, From the first equation:

$$\Delta t_0^l = \Delta t_0^u - \left({}^{1}\Delta_0 - {}^{0}\Delta_0\right)$$

Replace Δt_0^l in the MUX1 equation:

$$\Delta t_1^u = \Delta t_0^u + c_1\left(-\Delta t_0^u + \left[\Delta t_0^u - \left({}^{1}\Delta_0 - {}^{0}\Delta_0\right)\right]\right) + \epsilon_1^u + c_1\lambda_1^u$$

$$\boxed{\Delta t_1^u = \Delta t_0^u + c_1\left(-{}^{1}\Delta_0 + {}^{0}\Delta_0\right) + \epsilon_1^u + c_1\lambda_1^u}$$

This equation no longer explicitly depends on Δt_0^l
The delay at the i-th multiplexer is recursively defined as:

$$\boxed{\Delta t_i^u = \Delta t_{i-1}^u + c_i\left(-{}^{1}\Delta_{i-1} +{}^{0}\Delta_{i-1}\right) + \epsilon_i^u + c_i\lambda_i^u}$$

- Δt_{i-1}^u: Delay from the previous multiplexer $(i-1)$.
- c_i: Challenge bit for the i-th MUX ($c_i \neq 0$).
- ${}^{1}\Delta_{i-1} - {}^{0}\Delta_{i-1}$: Difference in upper-lower signal mismatches between PUF1 and PUF0 at MUX $i-1$.
- $\epsilon_i^u = {}^{1}p_i - {}^{0}p_i$: Intrinsic mismatch in parameter p_i.
- $\lambda_i^u = {}^{1}s_i - {}^{0}s_i - \epsilon_i^u$: Net mismatch between parameters s_i and p_i.

BASE CASE (MUX0):- The initial delay depends only on intrinsic and parameter mismatches:

$$\Delta t_0^u = \epsilon_0^u + c_0\lambda_0^u$$

For $i = 1$:

$$\Delta t_1^u = \Delta t_0^u + c_1\left(-{}^{1}\Delta_0 +{}^{0}\Delta_0\right) + \epsilon_1^u + c_1\lambda_1^u$$

Substitute $\Delta t_0^u = \epsilon_0^u + c_0\lambda_0^u$:

$$\Delta t_1^u = \epsilon_0^u + c_0\lambda_0^u + c_1\left(-{}^{1}\Delta_0 +{}^{0}\Delta_0\right) + \epsilon_1^u + c_1\lambda_1^u$$

For $i = 2$:

$$\Delta t_2^u = \Delta t_1^u + c_2\left(-{}^{1}\Delta_1 +{}^{0}\Delta_1\right) + \epsilon_2^u + c_2\lambda_2^u$$

Substitute Δt_1^u from above:

$$\Delta t_2^u = \epsilon_0^u + c_0\lambda_0^u + \epsilon_1^u + c_1\lambda_1^u + c_1\left(-{}^{1}\Delta_0 +{}^{0}\Delta_0\right) + c_2\left(-{}^{1}\Delta_1 +{}^{0}\Delta_1\right) + \epsilon_2^u + c_2\lambda_2^u$$

Generalized Delay Formula for MUX i

$$\boxed{\Delta t_i^u = \sum_{k=1}^{i} c_k\left(-{}^{1}\Delta_{k-1} + {}^{0}\Delta_{k-1}\right) + \sum_{k=0}^{i}\left(\epsilon_k^u + c_k\lambda_k^u\right)}$$

- $\sum_{k=1}^{i} c_k\left(-{}^{1}\Delta_{k-1} + {}^{0}\Delta_{k-1}\right)$: Cumulative signal adjustments.
- $\sum_{k=0}^{i}\left(\epsilon_k^u + c_k\lambda_k^u\right)$: Total intrinsic + parameter mismatches.

Definitions of Δ_{k-1} for simple arbiter PUF

For PUF1 ($n = 1$):

$$\boxed{{}^{1}\Delta_{k-1} = \sum_{j=0}^{k-1} \left({}^{1}w_j \cdot {}^{1}x_j\right) + {}^{1}\beta_{k-1}}$$

For PUF0 ($n = 0$):

$$\boxed{{}^{0}\Delta_{k-1} = \sum_{j=0}^{k-1} \left({}^{0}w_j \cdot {}^{0}x_j\right) + {}^{0}\beta_{k-1}}$$

Coefficient w_i:

$$\boxed{\begin{aligned} &{}^{n}w_0 = {}^{n}\alpha_0, \\ &{}^{n}w_i = {}^{n}\alpha_i + {}^{n}\beta_{i-1} \quad (i > 0). \end{aligned}}$$

Parameters α_i and β_i:

$$\boxed{\begin{aligned} {}^{n}\alpha_i &= \frac{1}{2}\left({}^{n}p_i - {}^{n}q_i + {}^{n}r_i - {}^{n}s_i\right), \\ {}^{n}\beta_i &= \frac{1}{2}\left({}^{n}p_i - {}^{n}q_i - {}^{n}r_i + {}^{n}s_i\right). \end{aligned}}$$

where $n = 0$ (PUF0) or $n = 1$ (PUF1).

Product x_j:

$$\boxed{x_j = \prod_{k=j}^{a-1} (1 - 2c_k) \quad \text{(Product of challenge bits from MUX } j \text{ to a -1).}}$$

After substituting the values the expression becomes:-

$$\boxed{\Delta t_i^u = \underbrace{\sum_{a=1}^{i} c_a \left(\sum_{m=0}^{a-1} x_m \Delta w_m + \Delta\beta_{a-1}\right)}_{\text{Signal Adjustments}} + \underbrace{\sum_{a=0}^{i} \left(\epsilon_a^u + c_a \lambda_a^u\right)}_{\text{Intrinsic + Parameter Mismatches}}}$$

Key Terms:

- $\Delta w_m = {}^{0}w_m - {}^{1}w_m$: Difference in coefficients w_m between PUF0 and PUF1.
- $\Delta\beta_{a-1} = {}^{0}\beta_{a-1} - {}^{1}\beta_{a-1}$: Difference in fixed parameters β_{a-1}.
- $\epsilon_a^u = {}^{1}p_a - {}^{0}p_a$: Intrinsic mismatch in parameter p_a.
- $\lambda_a^u = {}^{1}s_a - {}^{0}s_a - \epsilon_a^u$: Net mismatch between s_a and p_a.

Substituting $i = 1$ into the formula:

$$\begin{aligned} \Delta t_1^u &= c_1 \left(x_0 \Delta w_0 + \Delta\beta_0\right) + \epsilon_0^u + c_0 \lambda_0^u + \epsilon_1^u + c_1 \lambda_1^u, \\ &= \text{Matches recursive derivation.} \end{aligned}$$

Step 1: Substitute $x_m = \prod_{k=m}^{a-1}(1-2c_k)$
Replace x_m with the product term:

$$\Delta t_i^u = \underbrace{\sum_{a=1}^{i} c_a \left(\sum_{m=0}^{a-1} \left(\prod_{k=m}^{a-1} (1-2c_k) \right) \Delta w_m + \Delta\beta_{a-1} \right)}_{\text{Term 1}} + \underbrace{\sum_{a=0}^{i} \epsilon_a^u}_{\text{Term 2}} + \underbrace{\sum_{a=0}^{i} c_a \lambda_a^u}_{\text{Term 3}}$$

Step 2: Expand Term 1 Expand the nested sums in Term 1:

$$\text{Term 1} = \sum_{a=1}^{i} \sum_{m=0}^{a-1} c_a \left(\prod_{k=m}^{a-1} (1-2c_k) \right) \Delta w_m + \sum_{a=1}^{i} c_a \Delta\beta_{a-1}$$

Step 3: Separate c-Dependent Terms Group all terms involving c:

$$\begin{aligned}\Delta t_i^u = &\underbrace{\sum_{a=0}^{i} c_a \lambda_a^u + \sum_{a=1}^{i} c_a \Delta\beta_{a-1}}_{\text{Linear Terms}} \\ &+ \underbrace{\sum_{a=1}^{i} \sum_{m=0}^{a-1} c_a \left(\prod_{k=m}^{a-1} (1-2c_k) \right) \Delta w_m}_{\text{Product Terms}} + \underbrace{\sum_{a=0}^{i} \epsilon_a^u}_{b}\end{aligned}$$

Step 4: Define ψ^u Construct the vector ψ^u to include both linear and product terms:

$$\psi^u(c) = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_i \\ \sum_{m=0}^{0} c_1 \prod_{k=m}^{0}(1-2c_k) \\ \sum_{m=0}^{1} c_2 \prod_{k=m}^{1}(1-2c_k) \\ \vdots \\ \sum_{m=0}^{i-1} c_i \prod_{k=m}^{i-1}(1-2c_k) \end{bmatrix} \in \mathbb{R}^{(i+1)+\frac{i(i+1)}{2}}$$

where the product terms span all valid (a, m) pairs with $0 \leq m \leq a-1 \leq i-1$.

Step 5: Construct Matrix Q Matrix Q maps ψ^u to the delay formula:

$$Q = \left[\underbrace{\lambda_0^u}_{\text{For } c_0} \quad \underbrace{\Delta\beta_0 + \lambda_1^u}_{\text{For } c_1} \quad \cdots \quad \underbrace{\Delta\beta_{i-1} + \lambda_i^u}_{\text{For } c_i} \quad \underbrace{\Delta w_0}_{\substack{\text{For} \\ c_1(1-2c_0)}} \quad \underbrace{\Delta w_0}_{\substack{\text{For} \\ c_2(1-2c_0)(1-2c_1)}} \quad \underbrace{\Delta w_1}_{\substack{\text{For} \\ c_2(1-2c_1)}} \quad \cdots \right]$$

Each Δw_m corresponds to a product term in ψ^u.

Combine Q, ψ^u, and b:

$$\boxed{\Delta t_i^u = Q^u \cdot \psi^u(c) + b^u \quad \text{where} \quad \begin{cases} \psi^u(c) = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_i \\ \sum_{m=0}^{0} c_1 \prod_{k=m}^{0}(1-2c_k) \\ \sum_{m=0}^{1} c_2 \prod_{k=m}^{1}(1-2c_k) \\ \vdots \\ \sum_{m=0}^{i-1} c_i \prod_{k=m}^{i-1}(1-2c_k) \end{bmatrix} \in \mathbb{R}^{(i+1)+\frac{i(i+1)}{2}}, \\ Q^u = [\lambda_0^u \quad \Delta\beta_0 + \lambda_1^u \quad \cdots \quad \Delta\beta_{i-1} + \lambda_i^u \quad \Delta w_0 \quad \Delta w_0 \quad \Delta w_1 \quad \cdots], \\ b^u = \sum_{a=0}^{i} \epsilon_a^u. \end{cases}}$$

Step 6: Final Expression
Combine P^l, ψ^l, and b^l:

$$\boxed{\Delta t_i^l = P^l \cdot \psi^l(c) + b^l \quad \text{where} \quad \begin{cases} \psi^l(c) = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_i \\ \sum_{m=0}^{0} c_1 \prod_{k=m}^{0}(1-2c_k) \\ \sum_{m=0}^{1} c_2 \prod_{k=m}^{1}(1-2c_k) \\ \vdots \\ \sum_{m=0}^{i-1} c_i \prod_{k=m}^{i-1}(1-2c_k) \end{bmatrix} \in \mathbb{R}^{(i+1)+\frac{i(i+1)}{2}}, \\ P^l = \left[\lambda_0^l \quad \Delta\beta_0 + \lambda_1^l \quad \cdots \quad \Delta\beta_{i-1} + \lambda_i^l \quad \Delta w_0 \quad \Delta w_0 \quad \Delta w_1 \quad \cdots\right], \\ b^l = \sum_{a=0}^{i} \epsilon_a^l. \end{cases}}$$

Step 7: General XOR Formula for $K = 2$
The generalized XOR function is:

$$f = \frac{1 + (-1)^{K+1} \prod_i \operatorname{sign}(\mathbf{w}_i^\top \mathbf{x})}{2}$$

Substituting $K = 2$, we get:

$$f = \frac{1 - \operatorname{sign}(\Delta t_i^u) \cdot \operatorname{sign}(\Delta t_i^l)}{2}$$

Step 8: Matching to Linear Sign Function Form
We aim to express the XOR output as:

$$f = \frac{1 + \operatorname{sign}(\tilde{\mathbf{W}}^\top \tilde{\phi}(\mathbf{c}) + \tilde{b})}{2}$$

To match:

$$\frac{1 - \operatorname{sign}(\Delta t_i^u) \cdot \operatorname{sign}(\Delta t_i^l)}{2} = \frac{1 + \operatorname{sign}(-\Delta t_i^u \cdot \Delta t_i^l)}{2} = \frac{1 + \operatorname{sign}(\tilde{\mathbf{W}}^\top \tilde{\phi}(\mathbf{c}) + \tilde{b})}{2}$$

So we define:

$$\tilde{\mathbf{W}}^\top \tilde{\phi}(\mathbf{c}) + \tilde{b} = -\Delta t_i^u \cdot \Delta t_i^l$$

Hence:

$$\boxed{\tilde{\phi}(\mathbf{c}) = [\psi_j \psi_k, \psi_l]^\top, \quad \tilde{\mathbf{W}} = \text{constructed via expansion of } -(Q^u \cdot \psi(\mathbf{c}) + b^u)(P^l \cdot \psi(\mathbf{c}) + b^l), \quad \tilde{b} = -b^u b^l}$$

Final Result

$$f = \frac{1 + \text{sign}\left(-\Delta t_i^u \cdot \Delta t_i^l\right)}{2} = \frac{1 + \text{sign}\left(\tilde{\mathbf{W}}^\top \tilde{\phi}(\mathbf{c}) + \tilde{b}\right)}{2}$$

Computation of $\phi(\mathbf{c})$ from $\psi(\mathbf{c})$

Let $\psi(\mathbf{c}) = [\psi_1, \psi_2, \ldots, \psi_n]^T$. We compute the product:

$$\Delta t_i^u \cdot \Delta t_i^l = (Q^u \cdot \psi + b^u)(P^l \cdot \psi + b^l)$$

Expanding the terms:

$$\begin{aligned} \Delta t_i^u \cdot \Delta t_i^l &= (Q^u \cdot \psi)(P^l \cdot \psi) + b^u(P^l \cdot \psi) + b^l(Q^u \cdot \psi) + b^u b^l \\ &= \sum_{j=1}^{n} \sum_{k=1}^{n} Q_j^u P_k^l \psi_j \psi_k + b^u \sum_{k=1}^{n} P_k^l \psi_k + b^l \sum_{j=1}^{n} Q_j^u \psi_j + b^u b^l \end{aligned}$$

Therefore, the feature map $\tilde{\phi}(\mathbf{c})$ is:

$$\tilde{\phi}(\mathbf{c}) = \begin{bmatrix} \psi_1 \psi_1 \\ \psi_1 \psi_2 \\ \vdots \\ \psi_j \psi_k \\ \vdots \\ \psi_n \psi_n \\ \psi_1 \\ \vdots \\ \psi_n \end{bmatrix} \in \mathbb{R}^{[n(n+1)/2]+n}$$

2 DIMENSIONALITY

It includes all second-order pairwise products and all first-order terms of $\psi(\mathbf{c})$.

For $i = 7$:

$$\dim(\psi^u) = (i+1) + \frac{i(i+1)}{2} = 8 + \frac{7 \cdot 8}{2} = 8 + 28 = 36$$

$$\text{Let } n = \dim(\psi^u) = 36$$

$$\dim(\tilde{\phi}) = \frac{n(n+1)}{2} + n = \frac{36 \cdot 37}{2} + 36 = 666 + 36 = \boxed{702}$$

3 Kernel SVM

To replicate our model using a kernel SVM without explicit feature mapping, a polynomial kernel of degree 3 would be sufficient:

$$K(x, x') = (\gamma x^\top x' + r)^3$$

Choose $\gamma = 1$, $r = 0$. This kernel generates all 3rd, 2nd, and 1st-order interactions, matching our $\phi(x)$ structure.

4 Inversion Simple Arbiter

We have an arbitrary buffer with 4 delays P, Q, R, and S, where the direct delays are P and Q, and the cross delays are R and S. The task is to determine the non-negative delays given the models of the 64-bit buffers. In total, there are $64 \times 4 = 256$ variables, but only 65 equations (64 weights and one bias). This leaves us with 164 variables and only 65 equations.

The idea is to set all values of R and S to 0, and all values of Q to 0 except for q_{64}. This reduces the problem to 64 values of P and one value of Q, totaling 65 variables. Now, with 65 equations and 65 variables, we can solve for all 64 values of P and the 64^{th} value of Q.

However, some of these values may be negative, which violates the constraint that delays must be non-negative ($d \geq 0$). To address this we can do two things **1**. We add a constant to all P, Q, R, and S values to ensure they become non-negative. This constant is the absolute value of the largest negative number among the P values and q_{64}. **2**. We add a constant to only the P and Q values to ensure they become non-negative (others are already non-negative with 0 value). Similarly, this constant will also be the absolute value of the largest negative number among the P values and q_{64}.

Note: Since all the Ws have equal number of $+$ and $-$ sign delays (p, q, r, s, variable), so adding a constant i.e. $p_i + \epsilon_i$, $q_i + \epsilon_i$, $r_i + \eta_i$, $s_i + \eta_i$, in delays won't affect the Ws. Our model remains the same.

Here is the generalized model for the delay after passing through 64 responses:

$$\Delta_{64} = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_{64} \cdot x_{64} + b$$

The generalized weight w_i in terms of all delays is given by:

$$w_1 = \alpha_1$$

$$w_i = \alpha_i + \beta_{i-1} \quad (\text{for } i = 2, 3, \ldots, 64)$$

$$b = \beta_{64}$$

The delays are defined as:

$$d_i \stackrel{\text{def}}{=} (1 - 2c_i)$$

$$\alpha_i \stackrel{\text{def}}{=} (p_i - q_i + r_i - s_i)/2$$

$$\beta_i \stackrel{\text{def}}{=} (p_i - q_i - r_i + s_i)/2$$

Demonstration with a 2-bit Response

For simplicity, consider a 2-bit response where all Q and S values are set to 0, except for q_2 and all P_s. This leaves us with 3 equations and 3 variables, p_1, p_2, and q_2.

$$w_1 = \alpha_1 = \frac{(p_1 - q_1 + r_1 - s_1)}{2}$$

$$w_2 = \alpha_2 + \beta_1 = \frac{(p_2 - q_2 + r_2 - s_2)}{2} + \frac{(p_1 - q_1 - r_1 + s_1)}{2}$$

$$w_3 = b = \beta_2 = \frac{(p_2 - q_2 - r_2 + s_2)}{2}$$

The system can be represented in matrix form as:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ q_1 \\ q_2 \\ r_1 \\ r_2 \\ s_1 \\ s_2 \end{bmatrix}$$

Simplifying, we get:

$$w_1 = p_1$$

$$w_2 = p_2$$

$$w_3 - p_2 = q_2$$

Suppose we obtain $p_1 = -2$, $p_2 = 1$, and $q_2 = -3$. To ensure all values are non-negative, we can do two things as stated above.

1. Add 3 (the absolute value of the largest negative number, which is -3) to all P, Q, R, and S values. The final adjusted values will be:

p_1	p_2	q_1	q_2	r_1	r_2	s_1	s_2
1	4	0	3	3	3	3	3

2. Same as above, but just add 3 (the absolute value of the largest negative number, which is -3) only to all $P$and Q values as all other (R and S) values are already non-negative (kept 0 in the starting itself). Then the final adjusted values will be:

p_1	p_2	q_1	q_2	r_1	r_2	s_1	s_2
1	4	0	3	0	0	0	0

For our solution, we have chosen the second option.

5 CODE for ML-PUF

6 CODE for Invertor-PUF

7 EXPERIMENTS ON LinearSVC and Logistic Regression

SVM Loss Comparison

Loss Function	**Training Time (s)**	**Test Accuracy (%)**
Hinge Loss	2.33	95.75
Squared Hinge Loss	2.22	99.19

Table 1: Comparison of Hinge and Squared Hinge loss functions for SVM.

Effect of Regularization Parameter (C) on Accuracy and Time

C Value	**Model**	**Training Time (s)**	**Test Accuracy (%)**
0.0001 (Low)	LinearSVC	0.197	83.06
	LogisticRegression	0.269	69.19
0.01 (Medium)	LinearSVC	0.862	100.00
	LogisticRegression	0.782	96.75
10 (High)	LinearSVC	24.387	100.00
	LogisticRegression	0.934	100.00

Table 2: Effect of different regularization strengths (C values) on SVM and Logistic Regression.