Interpose PUF can be PAC Learned

Durba Chatterjee[1], Debdeep Mukhopadhyay[2], and Aritra Hazra[3]

[1] Indian Institute of Technology Kharagpur, India
durba@iitkgp.ac.in
[2] Indian Institute of Technology Kharagpur, India
debdeep@iitkgp.ac.in
[3] Indian Institute of Technology Kharagpur, India
aritrah@cse.iitkgp.ac.in

Abstract

In this work, we prove that Interpose PUF is learnable in the PAC model. First, we show that Interpose PUF can be approximated by a Linear Threshold Function (LTF), assuming the interpose bit to be random. We translate the randomness in the interpose bit to classification noise of the hypothesis. Using classification noise model, we prove that the resultant LTF can be learned with number of labelled examples (challenge response pairs) polynomial in the number of stages and PAC model parameters.

1 Introduction

The security of a cryptographic solution is dependent on the secrecy of the key employed. Storing cryptographic keys in Non-Volatile memories has been a major problem in hardware security. In such a situation, Physically Unclonable Function (PUF) is a preferable alternative to permanent key storage. PUFs are specialized circuits that give a different output (*response*) for a given input (*challenge*) on different chips. The challenge response behaviour of a PUF is dependent on the manufacturing variations of the chip. One of the most used PUF type is Arbiter PUF (APUF). It has been used in many cryptographic applications. The challenge response behaviour of an Arbiter PUF depends on the difference in the propagation delay of a signal traversing two identical paths.

However, APUF is vulnerable to machine learning attacks. This was followed by design and analysis of PUF constructions. Machine learning resistance is one of the important factors in the analysis. One of the recent PUF construction is the Interpose PUF, which claims to be resilient against all state of the art machine learning attacks. Most of the machine learning attacks against PUFs in the literature are ad-hoc and lack proper mathematical explanation. A formal mathematical framework for security assessment of PUFs has been proposed, for the first time in [3]. In this work, Arbiter PUF, XOR APUF, Ring Oscilltor PUF, Bistable Ring PUF have been proven to be vulnerable against modelling attacks. In this work, we perform a similar security assessment of Interpose PUF using the PAC framework. In [7], there is a discussion on PAC learning iPUF. However, in-depth analysis of the Percepron based attack is lacking.

2 Background

This section presents the background information required to understand the concept of PUFs, LTF, Perceptron algorithm and PAC model.

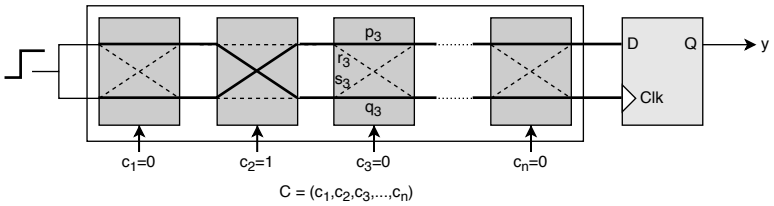

Figure 1: Block diagram of an Arbiter PUF

2.1 Physically Unclonable Functions

2.1.1 Arbiter PUF

Arbiter PUF is one of the first PUFs developed and henceforth has been used as a fundamental component is various other PUFs. It consists of a series of multiplexers followed by an arbiter. A signal passing in two identical paths through the multiplexers controlled by the challenge bits, reach the arbiter, which decides the output of the PUF, depending on whether the signal reaches the top or bottom input first. An n-bit APUF, consists of n multiplexers, each of which is controlled by a challenge bit. The schematic of an Arbiter PUF is shown in Figure 1.

Let $c \in \{-1, 1\}^n$ be a challenge and $y \in \{-1, 1\}$ be the response where n be the number of switches in the APUF. Let $c_i \in \{-1, 1\}$ be the i^{th} challenge bit. The total delay at the $(i+1)^{th}$ stage can be given as follows:

$$\delta_t(i+1) = \frac{1+c_{i+1}}{2}(p_{i+1} + \delta_t(i)) + \frac{1-c_{i+1}}{2}(s_{i+1} + \delta_b(i)) \tag{1}$$

$$\delta_b(i+1) = \frac{1+c_{i+1}}{2}(q_{i+1} + \delta_b(i)) + \frac{1-c_{i+1}}{2}(r_{i+1} + \delta_t(i)) \tag{2}$$

$$\Delta(i+1) = c_{i+1}.\Delta(i) + \alpha_{i+1}.c_{i+1} + b_{i+1} \tag{3}$$

$$\alpha_i = \frac{(p_i - q_i) + (r_i - s_i)}{2} \quad \text{and} \quad \beta_i = \frac{(p_i - q_i) - (r_i - s_i)}{2} \tag{4}$$

$$p_k = \prod_{i=k+1}^{n} c_i \quad k = 0, 1 \cdots, n-1 \quad p_n = 1 \tag{5}$$

$$\begin{aligned} \Delta(n) &= \alpha_1 p_0 + (\alpha_2 + \beta_1)p_1 + \cdots + (\alpha_n + \beta_{n-1})p_{n-1} + \beta_n p_n \\ &= \langle \mathbf{w}, \phi \rangle \end{aligned} \tag{6}$$

The APUF output is given by $y = sign(\Delta(n))$.

2.1.2 XOR Arbiter PUF

Due to the existence of a linear additive model, APUF is vulnerable to machine learning attacks. XOR Arbiter PUF [11] was introduced to increase the robustness against machine learning attacks. A k-XOR Arbiter PUF also known as k-XOR PUF consists of k APUFs of equal length. Each of the constituent APUF is given the same challenge as input and their responses are combined by an XOR gate. The schematic of a 3-XOR PUF is shown in Figure 2. However, this construction has also been found to be vulnerable against modelling attacks [8].

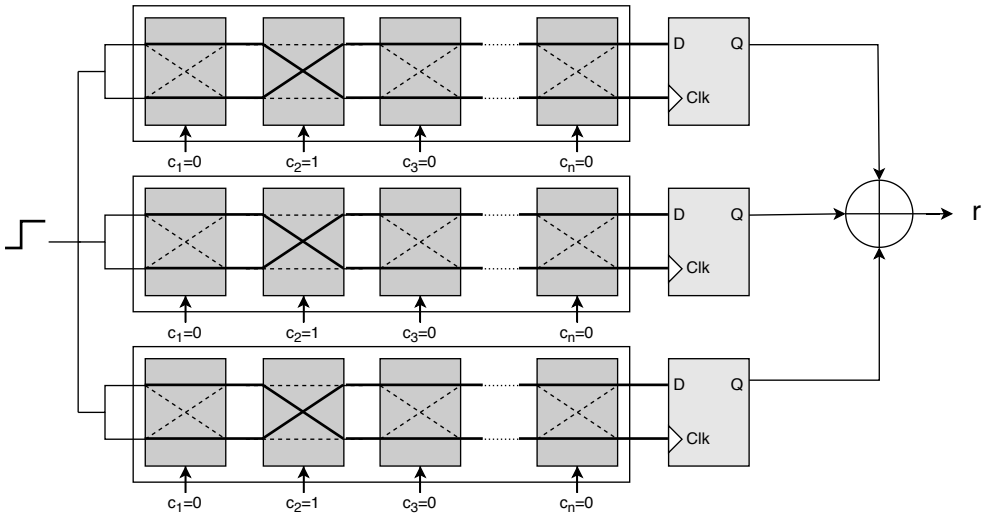

Figure 2: Block diagram of a 3-XOR Arbiter PUF

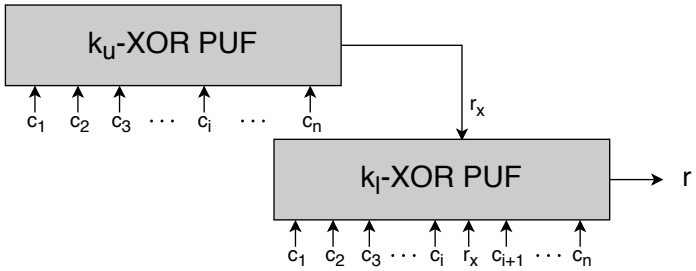

Figure 3: Block diagram of (k_u, k_l)-Interpose PUF

2.1.3 Interpose PUF

Interpose PUF [7] is a strong PUF constructed using Arbiter PUFs. The PUF construction claims to mitigate the classical and reliability based machine learning attacks. An Interpose PUF has a layered construction. A (k_u, k_l)-iPUF takes an n-bit challenge as input and returns a 1-bit output. The k_u-XOR APUF in the upper layer takes the n-bit challenge and returns a 1-bit response. The output of the k_u-XOR PUF is interposed at a given position in the input of the second k_l-XOR APUF. The k_l-XOR APUF takes an $(n+1)$ bit challenge and returns a 1-bit response which is the final response of the iPUF. The schematic of (k_u, k_l)-iPUF is shown in Figure 3.

2.2 Linear threshold functions and Perceptron algorithm

A Linear Threshold Function $h : \mathbb{R}^n \to \{0, 1\}$ is given by:

$$h = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} (w[i].\phi[i]) \geq \theta \\ 0, & \text{if } \sum_{i=1}^{n} (w[i].\phi[i]) < \theta \end{cases} \tag{7}$$

where $\phi \in \mathbb{R}^n$ is the input vector and $\mathbf{w}$ represents the weight vector. The sets of positive and negative examples of h form two halfspaces S^0 and S^1 where $S^1 = \{\phi \in \mathbb{R}^n | \sum_{i=1}^{n}(w[i].\phi[i]) \geq \theta\}$ and $S^0 = \{\phi \in \mathbb{R}^n | \sum_{i=1}^{n}(w[i].\phi[i]) < \theta\}$. Mapping $\{0,1\} \rightarrow \{1,-1\}$, including the constant in the weight vector $\mathbf{w}$ and appending 1 to the input vector ϕ, we get

$$h = sign(\mathbf{w}.\phi)$$

where $\mathbf{w} = (w_1, w_2, \cdots, w_n, \theta)$ and $\phi = (\phi[1], \cdots, \phi[n], 1)$. Decision hyperplane is given by $\mathcal{P} : \mathbf{w}.\phi = 0$

Perceptron Algorithm
Perceptron algorithm is an online algorithm used to learn LTF efficiently. The algorithm takes a set of r labelled examples $\langle(\phi_1, y_1), (\phi_2, y_2), \cdots, (\phi_r, y_r)\rangle$ and outputs a vector $\mathbf{w}$. It begins with a zero vector ($\mathbf{w}_0 = (w_0[1], w_0[2], \cdots, w_0[n], \theta_0) = (0, \cdots, 0)$) and updates the vector when there is a mismatch between the actual and the predicted label. Let $\mathbf{w}_{j-1}$ be the weight vector before the j^{th} mistake. The updated vector $\mathbf{w}_j$ is computed as

$$\mathbf{w}_j[i] = \begin{cases} \mathbf{w}_{j-1}[i] + y_j.\phi_j[i] & 1 \leq i \leq n \\ \theta_j - y_j & i = n+1 \end{cases} \tag{8}$$

The convergence theorem of the Perceptron algorithm gives an upper bound of the error that can occur during the execution of Perceptron algorithm.

Convergence theorem of the Perceptron Algorithm:
Let $\langle(\phi_1, y_1), (\phi_2, y_2), \cdots, (\phi_r, y_r)\rangle$ be a sequence of labelled examples with $\|x_i\| \leq R$. Let $\mathbf{w}*$ be the solution vector with $\|\mathbf{w}^*\| = 1$ and let $\gamma > 0$. The deviation of each example is defined as $d_i = max\{0, \gamma - y_i(\mathbf{w}^*.\phi_i)\}$, and $D = \sqrt{\sum_{i=1}^{r} d_i^2}$. The number of mistakes of the online Perceptron algorithm on this sequence is bounded by

$$N_{mis} = \Big(\frac{R+D}{\gamma}\Big)^2$$

(Please refer [2] for details.)

2.3 PAC Model

The Probably Approximately Correct or PAC model of learning is a general model which enables us to formally analyse machine learning algorithms. It consists of a learning algorithm ($\mathcal{A}$), which is provided with a set of examples picked from the input space $\mathcal{X}$ as per distribution $\mathcal{D}$ and labelled using the target function f. The objective of the algorithm is to deliver an approximately correct hypothesis with high probability. It can be formally stated as follows:

Let $\mathcal{C}_n$ be a concept class defined over an instance space $\mathcal{X}_n = \{0,1\}^n$ and let $\mathcal{X} = \cup_{n\geq 1}\mathcal{X}_n$ and $\mathcal{C} = \cup_{n\geq 1}\mathcal{C}_n$. Let f be the target function in $\mathcal{C}_n$. Let $\mathcal{H}_n$ be the hypothesis class and $\mathcal{H} = \cup_{n\geq 1}\mathcal{H}_n$. The concept class $\mathcal{C}_n$ is said to be PAC Learnable if there exists a learning algorithm $\mathcal{A}$, polynomial $p(.,.,.)$ and values ϵ and δ with the following property: For every $\epsilon, \delta \in (0,1)^2$, for every distribution $\mathcal{D}$ over $\mathcal{X}_n$ and every target concept $f \in \mathcal{C}_n$, when $\mathcal{A}$ is provided with $p(n, 1/\epsilon, 1/\delta)$ independent examples drawn with respect to $\mathcal{D}$ and labelled according to f, then with probability atleast $1-\delta$ the algorithm $\mathcal{A}$ returns a hypothesis $h \in \mathcal{H}_n$ such that $error(h) \leq \epsilon$. The smallest polynomial p satisfying this condition is the sample complexity of

$\mathcal{A}$. The concept class $\mathcal{C}$ is said to be properly PAC Learnable if $\mathcal{C} = \mathcal{H}$. When $\mathcal{C} \neq \mathcal{H}$, $\mathcal{C}$ is known as agnostic PAC Learnable. The error of the hypothesis h with respect to target f is defined as $error(h) = \sum_{x \in h \triangle f} \mathcal{D}(x)$ where $\triangle$ denotes the symmetric difference.

Conversion from online to PAC Learning model: There are various conversion mechanisms to convert an online algorithm to a PAC Learning algorithm [9]. We use the method used in [6] as it is asymptotically the most efficient. The steps are as follows:

1. A sequence of labelled examples obtained from Oracle $EX()$ is fed to the online algorithm $\mathcal{A}_{on}$.
2. Hypotheses generated by $\mathcal{A}_{on}$ are stored.
3. A new sequence of labelled examples is obtained from $EX()$ which is used to calculate the error rate of the hypotheses stored. The hypothesis with the lowest error rate is the outputted.

The sample complexity of the PAC learning algorithm is related to the mistake bound N_{mis} by the following theorem proved in [6].

Theorem: Let $\mathcal{A}_{on}$ be an online algorithm that updates its hypothesis only when the predicted and received label differ. The total number of calls that the PAC algorithm $\mathcal{A}$ makes to the Oracle is $\mathcal{O}(1/\epsilon(log(1/\delta) + N_{mis}))$.
Therefore, the following holds: (Refer corollary 1 in [4])

Corollary: Let $\mathcal{C}_n$ be the class of LTF over $\{0,1\}^n$ such that $\mathbf{w}_i \in \mathbb{Z}$ and $\sum_{i=1}^{n} |\mathbf{w}[i]| \leq U$, then the online Perceptron algorithm can be converted to a PAC learning algorithm running in time $poly(n, U, 1/\epsilon, 1/\delta)$.

Note that the weight vector can be converted to an integer vector by using the delay discretization technique given in [5]. The delay discretization step states that the delay values of an Arbiter PUF can be mapped to in integer value in the range $[-m, m]$ where $m = \lceil 6\sigma/\kappa \rceil, \kappa$ is the precision of the arbiter and σ is the variance of the delay distribution. Therefore $w_i \in [-2m, 2m]$.

3 PAC Learning of Interpose PUF

A (k_u, k_l)-iPUF takes an n-bit challenge as input and returns a 1-bit output. It comprises of 2 XOR PUFs in two layers. The k_u-XOR APUF in the upper layer takes the n-bit challenge and returns a 1-bit response. The output of the k_u-XOR PUF is given is interposed at a given position in the input of the second k_l-XOR APUF. The k_l-XOR APUF takes an $(n+1)$ bit challenge and returns a 1-bit response which is the final response of the iPUF. Let the interpose bit position be t. Due to the unknown interpose bit in the second XOR PUF, it has proven to be resistant against modelling attacks.

If the interpose bit and the bit position is known, then an n-bit (k_u, k_l)-iPUF reduces to an $(n+1)$-bit k_l-XOR APUF. Considering the interpose bit to be random, an instance of iPUF can be considered to be an XOR PUF which takes a $(n+1)$-bit challenge and gives a one bit response. Since both the constituent XOR PUFs are given the same challenge, n out of $n+1$ bits are known and only one bit needs to be chosen at random. XOR PUFs have been proven

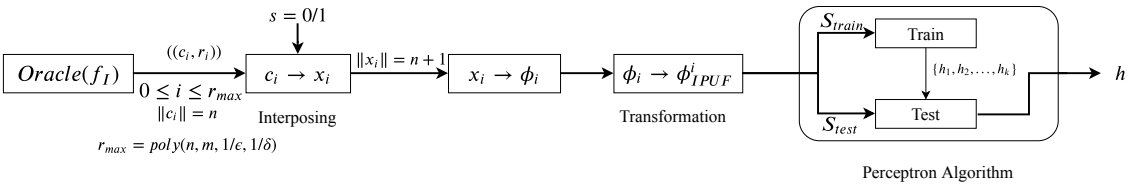

Figure 4: Schematic for PAC Learning of iPUF represented as an LTF.

to be learnable in the PAC model [4] under LTF representation. Thus, we use LTF to represent the class of Interpose-PUF, by assuming the interpose bit to be random.

First, we represent an (k_u, k_l)-iPUF as a Linear Threshold Function by fixing an interpose bit position. The Perceptron algorithm used to learn LTF and its conversion to the PAC model has been presented in the previous section. Next, we present the PAC Learning results for XOR PUFs obtained in [4]. We then use the theoretical bias calculation given in [10] to explain the dependence of the iPUF output on the interpose bit. Next we translate the randomness in interpose bit to classification noise model and estimate the error in PUF response. We calculate the probability of disagreement between the hypothesis (h) obtained from the PAC learning algorithm ($\mathcal{A}$) and the sample produced the target hypothesis (f_I). Finally, we calculate the number of mistakes allowed for the Perceptron algorithm and the sample complexity (number of challenge response pairs) to PAC-learn (k_u, k_l)-iPUF.

3.1 LTF Representation of Interpose PUF

We adopt the LTF representation of XOR PUFs [4], to represent the Interpose PUF. The lower XOR PUF in an n-bit iPUF takes an $(n+1)$-bit challenge, which comprises of the n-bit challenge given as input to the iPUF and an additional interpose bit which is the response of the upper XOR PUF. According to the linear additive model of APUF, we have

$$\Delta(n+1) = \langle \mathbf{w}^T.\phi \rangle$$

where $\mathbf{w}$ is the weight vector of length $n+2$ (including the bias) and ϕ is the encoded challenge vector or parity vector. Assuming $y \in \{-1, 1\}$, the output of an arbiter is given by

$$f_{APUF} = sgn(\Delta(n+1))$$

Assuming the interpose bit and the bit position is known, (k_u, k_l)-iPUF can be represented as follows

$$\begin{aligned} f_{IPUF} &= \prod_{j=1}^{k_l} sgn(\mathbf{w}^T.\phi) = sgn\Big(\bigotimes_{j=1}^{k_l} \mathbf{w}^T . \bigotimes_{j=1}^{k_l} \phi_j\Big) \\ &= sgn\Big(\mathbf{w}^T_{IPUF}.\phi_{IPUF}\Big) \end{aligned} \tag{9}$$

where k_l is the number of Arbiter chains in the lower XOR PUF, $\mathbf{w}_{IPUF} = \otimes_{j=1}^{k_l} \mathbf{w}_j^T$ is the tensor product of the weight vectors and $\phi_{IPUF} = \otimes_{j=1}^{k_l} \phi_j$ is the tensor product of the parity vector.

Please refer Section 3.1 in [4].

PAC Learning of XOR APUF with Perceptron: The Perceptron algorithm can be applied to learn k-XOR APUF after transforming the $(n+1)$- dimensional vectors to $(n+1)^k$ dimension (refer Equation 9) as shown in [4].

From [4], we get the following result:

Result: For a class of k-XOR PUF over the instance space $X_n = \{0,1\}^{(n+1)^k}$ represented by the class of linear threshold functions such that $w_i \in \mathbb{Z}$ and $\sum_{i=1}^{(n+1)^k} |w_i| \leq 2m(n+1)^k$, the Perceptron-based algorithm running in time $poly((n+1)^k, 4m^2, 1/\epsilon, 1/\delta)$ can PAC learn an XOR PUF by calling the Oracle at most $\mathcal{O}(log(1/\delta)/\epsilon + 4m^2(n+1)^k/\epsilon^3)$ times.

In the following sections, we estimate the classification noise to the random interpose bit and use the method given in [1] which shows that Perceptron algorithm can handle classification noise to prove that iPUF is PAC learnable.

3.2 Dependence of Response on the interpose bit

In this section, we calculate the dependence of iPUF response on the interpose bit to understand the impact of setting a random interpose bit. Assuming the interpose bit position to be t, we estimate the dependence of iPUF output on the interpose bit by calculating the probability with which the output of an iPUF will flip on flipping the interpose bit. This probability is obtained from the bias of the PUF response. Since the iPUF can be abstracted as an XOR PUF with an extra (unknown) bit, we calculate the bias of an k-XOR PUF which takes an $(n+1)$-bit challenge as given in [10]. The bias of a k-XOR PUF is calculated from the bias of its constituent APUFs.

Let x and $\hat{x}$ be two $(n+1)$-bit challenges which differ in the t^{th} position. Let z and $\hat{z}$ be the corresponding responses. The bias of an APUF is given by $Pr[z = \hat{z}]$. From [10], we get

$$Pr[z = \hat{z}] = 1 - \frac{2}{\pi} \tan^{-1} \sqrt{\frac{2t-1}{2(n+1)-2t+1}} \tag{10}$$

The XOR response is given by $r = \bigoplus_{i=1}^{k} sign(\Delta_i(n))$.

The bias on a k-XOR PUF can be calculated using Equation 10 and Piling Lemma.

The Piling Lemma states the if $Pr[X_i = 0] = p_i = \frac{1}{2} + \epsilon_i$ for $i = 1, \cdots, k$, then $Pr[\bigoplus_{i=1}^{k} X_i = 0] = \frac{1}{2} + 2^{k-1} \prod_{i=1}^{k} \epsilon_i$.

Thus the bias of k-XOR PUF is given by:

$$Pr[z = \hat{z}] = \frac{1}{2} + 2^{k-1}\epsilon^k = \eta$$

where

$$\epsilon = \frac{1}{2} - \frac{2}{\pi} \tan^{-1} \sqrt{\frac{2t-1}{2(n+1)-2t+1}}$$

Consequently, $Pr[z \neq \hat{z}] = 1 - \eta$.

3.3 Estimation of classification noise due to random interpose bit

Let h be the hypothesis output by the Perceptron-based algorithm and f_I be the oracle used for generating challenge response pairs. Let (c, r) be a challenge response pair provided by the oracle f_I, where $c = (c_1, c_2, \cdots, c_n) \in \{-1, 1\}^n$ and $r \in \{-1, 1\}$. The lower XOR APUF takes an $(n+1)$-bit challenge. Let it be denoted by $x = (c_1, c_2, \cdots, c_{t-1}, s, c_t, \cdots, c_n)$ where s is the interpose bit. As per the Perceptron algorithm, the hypothesis (h) is updated only when the predicted response doesn't match with the actual response.
Here, h is a LTF and is represented by a weight vector $\mathbf{w} = (w_1, w_2, \cdots, w_{t-1}, w_s, w_t, \cdots, w_n, w_{n+1})$ of length $n+2$. Given challenge c, first the input x is generated as $x = (c_1, c_2, \cdots, c_t, s, c_{t+1}, \cdots, c_n)$ by choosing $s \in \{-1, 1\}$ randomly. Then the corresponding parity vector ϕ is calculated as $\phi[i] = \prod_{j=i}^{n+1} x[j]$. Finally the output of the hypothesis h is given by $y = sign(\mathbf{w}.\phi)$. Let C_h be the set of challenges for which the hypothesis returns 1 as output and C_{f_I} be the set of challenges for which f_I returns 1 as output. then d_h can be defined as follows

$$d_h = \mathcal{D}(C_h \triangle C_{f_I}) = \sum_{c \in C_h \triangle C_{f_I}} Pr_{\mathcal{D}}[c]$$

where $C_h \triangle C_{f_I}$ denotes the set of challenges for which the output of the Oracle does not match with the output of the hypothesis, irrespective of the value of s. As per PAC model, $d_h \leq \epsilon$.

Thus the probability that an example (c, r) produced by f_I disagrees with hypothesis h is calculated as follows:

$$\begin{aligned} Pr_{c\sim\mathcal{D}}[h(x) \neq f_I(c)] = p &= Pr_{c\sim\mathcal{D}}[c \in C_h \triangle C_{f_I}](Pr[s \text{ doesn't affect } r]) \\ &\quad + Pr_{c\sim\mathcal{D}}[c \notin C_h \triangle C_{f_I}]\big(Pr[s \text{ affects } r].Pr[s \text{ is incorrect}]\big) \\ &= d_h.\eta + ((1-d_h)(1-\eta).\frac{1}{2}) \\ &= \frac{(1-\eta)}{2} + \frac{d_h(3\eta-1)}{2} \\ &\leq \frac{(1-\eta)}{2} + \frac{\epsilon(3\eta-1)}{2} \qquad [d_h \leq \epsilon] \end{aligned} \tag{11}$$

As shown above, the probability that the hypothesis output disagrees with the output of f_I depends on two factors: i) The dependence of hypothesis output on the interpose bit and ii) the challenge lying in $C_h \triangle C_{f_I}$. Thus the there are two possibilities: i) The challenge lies in $C_h \triangle C_{f_I}$ implies that the hypothesis and the target function will differ on that challenge and the interpose bit does not affect the final response. ii) The hypothesis and the target function agree on a challenge and due to the incorrect interpose bit, the response gets flipped.

When hypothesis h becomes equal to the target hypothesis f, we have $(\epsilon = 0) \implies p = (1-\eta)/2$, since error will occur only due to the change in response as a result of random selection of interpose bit. Since $1/2 \leq \eta \leq 1$, for any hypothesis h, the error is atleast $(1-\eta)/2$. Now for a ϵ-bad hypothesis (g), $\mathcal{D}(g \triangle f_I) \geq \epsilon$, thus

$$p \geq \frac{(1-\eta)}{2} + \frac{\epsilon(3\eta-1)}{2}$$

Thus we have a separation of atleast $\epsilon(3\eta-1)/2$ between the disagreement rates of correct and ϵ-bad hypothesis [1].

3.4 Sample complexity

The maximum number of mistakes that can be made by the Perceptron algorithm is polynomial in the separation $(\epsilon(3\eta-1)/2)$ and is given by

$$N_{mis} = \Big(\frac{2R}{\epsilon(3\eta-1)}\Big)^2$$

where $\|\phi\| \leq R$ and $\|\mathbf{w^*}\| = 1$ where $\mathbf{w^*}$ is the weight vector corresponding to hypothesis output by the Perceptron algorithm after feeding r CRP. (refer Theorem 3 in [4]).

Theorem 3[[4]] Consider r labeled examples $< (\phi_1, r_1), \cdots (\phi_i, r_i), \cdots (\phi_r, r_r) >$ which are fed into the Perceptron algorithm and let $\|\phi\| \leq R$. In case of a noisy labels (due to random interpose bit), let $\mathbf{w^*}$ be the solution vector with $\|\mathbf{w^*}\| = 1$, and $(\mathbf{w^*}.\phi_i).r_i \geq \epsilon(3\eta-1)/2 > 0$, then

$$N_{mis} = \Big(\frac{2R}{\epsilon(3\eta-1)}\Big)^2$$

We know that $\|\phi\| \leq (n+2)^k$. Therefore,

$$N_{mis} = \frac{4(n+2)^k}{\epsilon^2(3\eta-1)^2}$$

where

$$\eta = \frac{1}{2} + 2^{k-1}\Big(\frac{1}{2} - \frac{2}{\pi}\tan^{-1}\Big(\sqrt{\frac{2t-1}{2(n+1)-2t+1}}\Big)\Big)^k$$

If N_{mis} is the number of mistakes made by the learning algorithm, and the learning algorithm updates its hypothesis only when the predicted response and actual response differs, then the total number of calls that the learning algorithm makes to the Oracle (iPUF in our case) is $\mathcal{O}(1/\epsilon(log(1/\delta) + N_{mis}))$ [6].

Therefore, number of CRPs required

$$\begin{aligned} &= \mathcal{O}\Big(\frac{1}{\epsilon'}\Big(log(\frac{1}{\delta}) + N_{mis}\Big)\Big) \\ &= \mathcal{O}\Big(\frac{2log(1/\delta)}{\epsilon(3\eta-1)} + \frac{32m^2(n+2)^k}{\epsilon^3(3\eta-1)^3}\Big) \end{aligned} \tag{12}$$

The sample complexity is $poly(n, m, 1/\epsilon, 1/\delta)$. It is to be noted that in our calculation, we have fixed the interpose bit position. However, varying the interpose bit position across all the stages and adding the number of challenge response pairs required, the sample complexity will still remain $poly(n, m, 1/\epsilon, 1/\delta)$. This proves that Interpose PUF is PAC Learnable.

3.5 Perceptron algorithm for learning LTF with random interpose bit

The Perceptron based PAC algorithm takes a sample S comprising of r_{max} CRPs and splits it into training sample S_{train} and test sample S_{test}. In the training phase, the algorithm takes S_{train} CRPS and the number of APUFs in the lower XOR APUF as input and outputs a set of weight vector $\mathbf{w}_i$, each corresponding to one of the hypothesis generated during the execution of the Perceptron algorithm. Each of these hypothesis is then tested with the CRPs in S_{test} and the hypothesis with the least error is given as output. The Oracle f_I gives challenges of length n, which is then transformed to obtain $(n+2)^k$ dimensional vector. The training and testing phase of the Perceptron-based algorithm is given in Algorithm 1 and 2.

Algorithm 1: Training phase of learning algorithm

Input: Labeled examples $\langle (C_i, r_i) \rangle, 0 \leq i \leq |S_{train}|$
Number of APUFs in lower XOR APUF: k
Output: Weight vectors: v
Initialize $j \leftarrow 1, \mathbf{w} \leftarrow 0$
Initialize $v \leftarrow \{\}$
for $i = 1$ *to* $|S_{train}|$ **do**
  $s \xleftarrow{R} \{0, 1\}$
  $x_i \leftarrow \langle c_{i,1}, \cdots, c_{i,t}, s, c_{i,t+1}, \cdots, c_{i,n} \rangle$
  $\phi_i \leftarrow \mathsf{parity\_generation}(x_i)$
  $\hat{y}_i \leftarrow \mathsf{sign}(\mathbf{w}_j.\phi_i)$
  if $\hat{y}_i = r_i$ **then**
    $s_i = s$
  end
  else
    $x_i \leftarrow \langle c_{i,1}, \cdots, c_{i,t}, -s, c_{i,t+1}, \cdots, c_{i,n} \rangle$
    $\phi_i^{\oplus s} \leftarrow \mathsf{parity\_generation}(x_i^{\oplus s})$
    $\hat{y}_i \leftarrow \mathsf{sign}(\mathbf{w}_j.\phi_i^{\oplus s})$
    if $\hat{y}_i = r_i$ **then**
      $s_i = -s$
    end
    else
      $v \leftarrow v \cup \mathbf{w}_j$
      $\mathbf{w}_{j+1} \leftarrow \mathbf{w}_j + \phi_i.r_i$
      $j \leftarrow j + 1$
    end
  end
end
Return v

4 Conclusion

We proved that Interpose PUF is efficiently learned in the PAC model for various values of accuracy and confidence. To this end, we presented an approximate representation of an Interpose PUF as a Linear Threshold function with a random input bit. We translated the randomness in the interpose bit to classification noise. We showed that the classification noise is bounded by 1/2. Finally we showed that the original perceptron algorithm that can be used in this noisy setting to efficiently learn an LTF and proved that the sample complexity is polynomial in the number of stages (n), maximum propagation delay (m) and the PAC model parameters.

References

[1] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[2] Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[3] Fatemeh Ganji. *On the Learnability of Physically Unclonable Functions*. Springer, 2018.

Algorithm 2: Testing phase of learning algorithm

Input: Labeled examples: $\langle (C_i, r_i) \rangle, 0 \leq i \leq |S_{test}|$
Weight vectors: v $= (\mathbf{w}_1, \cdots, \mathbf{w}_k)$
Output: Weight vector: $\mathbf{w}$
Initialize $mis \leftarrow (0)_k$
for $j = 1$ *to* k **do**
  for $i = 1$ *to* $|S_{test}|$ **do**
    $s \xleftarrow{R} \{0, 1\}$
    $x_i \leftarrow \langle c_{i,1}, \cdots, c_{i,t}, s, c_{i,t+1}, \cdots, c_{i,n} \rangle$
    $\phi_i \leftarrow \mathsf{parity\_generation}(x_i)$
    $\hat{y}_i \leftarrow \mathsf{sign}(\mathbf{w}_j.\phi_i)$
    if $\hat{y}_i \neq r_i$ **then**
      $mis[j] \leftarrow mis[j] + 1$
    end
  end
end
$J \leftarrow \underset{j}{\text{argmin}}(mis)$
Return $v[J]$

[4] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. Why attackers win: on the learnability of xor arbiter pufs. In *International Conference on Trust and Trustworthy Computing*, pages 22–39. Springer, 2015.

[5] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. Pac learning of arbiter pufs. *Journal of Cryptographic Engineering*, 6(3):249–258, 2016.

[6] Nick Littlestone. From on-line to batch learning. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, COLT '89, page 269–284, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[7] Phuong Ha Nguyen, Durga Prasad Sahoo, Chenglu Jin, Kaleel Mahmood, Ulrich Rührmair, and Marten van Dijk. The interpose puf: Secure puf design against state-of-the-art machine learning attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 243–290, 2019.

[8] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 237–249, 2010.

[9] Rocco Anthony Servedio. *Efficient algorithms in computational learning theory.* 2001.

[10] Akhilesh Anilkumar Siddhanti, Srinivasu Bodapati, Anupam Chattopadhyay, Subhamoy Maitra, Dibyendu Roy, and Pantelimon Stanica. Analysis of the strict avalanche criterion in variants of arbiter-based physically unclonable functions. In *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*, volume 11898 of *Lecture Notes in Computer Science*, pages 556–577. Springer, 2019.

[11] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14. IEEE, 2007.