PAC Learnability of iPUF Variants

Durba Chatterjee, Debdeep Mukhopadhyay, and Aritra Hazra

Indian Institute of Technology Kharagpur, India
durba@iitkgp.ac.in, debdeep@iitkgp.ac.in, aritrah@cse.iitkgp.ac.in

Abstract

Interpose PUF (iPUF) is a strong PUF construction that was shown to be vulnerable against empirical machine learning as well as PAC learning attacks. In this work, we extend the PAC Learning results of Interpose PUF to prove that the variants of iPUF are also learnable in the PAC model under the Linear Threshold Function representation class.

1 Introduction

Interpose PUF (iPUF) [7], proposed in CHES 2019 claimed to be secure against state-of-the-art machine learning (ML) attacks. However in CHES 2020 [9], it was demonstrated to be vulnerable against a targetted attack using Logistic Regression (LR) algorithm that models each layer PUF one at a time. The attack was followed by the proposition of some variants of iPUF, namely Domino-iPUF, XOR-Domino-iPUF, Tree-iPUF and XOR-Cascaded-iPUF, which were also shown to be vulnerable against empirical ML attacks.The first provable ML attack on iPUF was reported in [1], wherein iPUF construction was proven to be PAC Learnable, by assuming the interpose bit to be random, thereby eliminating the upper layer PUFs. However no results were presented for the iPUF compositions. In this work, we evaluate the learnability of these constructions in the PAC learning framework.

2 Background

This section briefly describes the concept of PUFs, LTF, Perceptron algorithm and PAC model.

2.1 Physically Unclonable Functions

Arbiter PUF and XOR 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.

The challenge-response behaviour of an APUF can be represented using a linear delay model [5]. 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 difference at the end of $(n)^{th}$ stage can be given by :

$$\Delta(n) = \langle \mathbf{w}, \phi \rangle \tag{1}$$

where $\mathbf{w}$ denotes the weight vector comprising of the propagation delays and ϕ is the parity vector or the transformed challenge vector. The APUF output is given by $y = sign(\Delta(n))$.

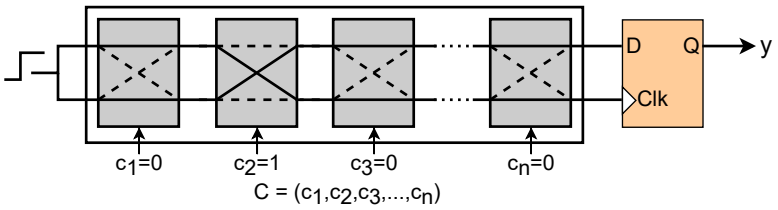


Figure 1: Block diagram of an Arbiter PUF

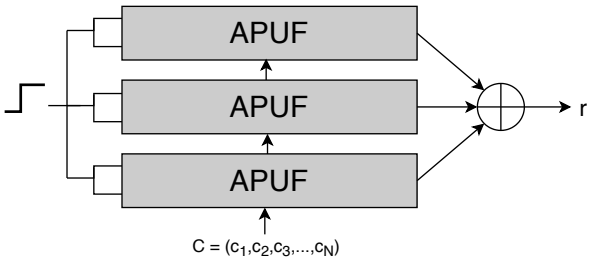


Figure 2: Block diagram of a 3-XOR Arbiter PUF

XOR Arbiter PUF is a composition comprising of multiple APUFs where each of the constituent APUFs is fed the same challenge and their responses are combined by an XOR gate.

Interpose PUF (iPUF)

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. 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:

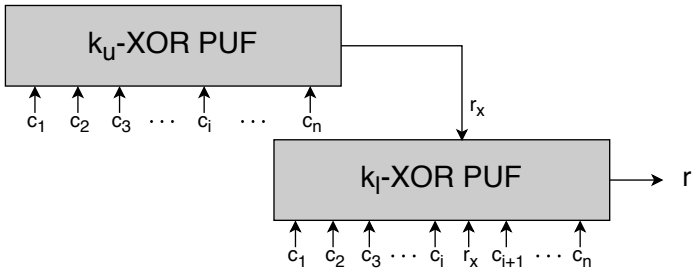


Figure 3: Block diagram of (k_u, k_l)-Interpose PUF

$$h = \begin{cases} 1, & \text{if } \sum_{i=1}^{n}(w[i].\phi[i]) \geq \theta \\ 0, & \text{if } \sum_{i=1}^{n}(w[i].\phi[i]) < \theta \end{cases} \tag{2}$$

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

$$\mathbf{w}_j[i] = \begin{cases} \mathbf{w}_{j-1}[i] + y_j.\phi_j[i] & 1 \leq i \leq n \\ \theta_j - y_j & i = n+1 \end{cases} \tag{3}$$

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

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

(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 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: Among various conversion mechanisms 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 [3])

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 [4]. 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 Variants

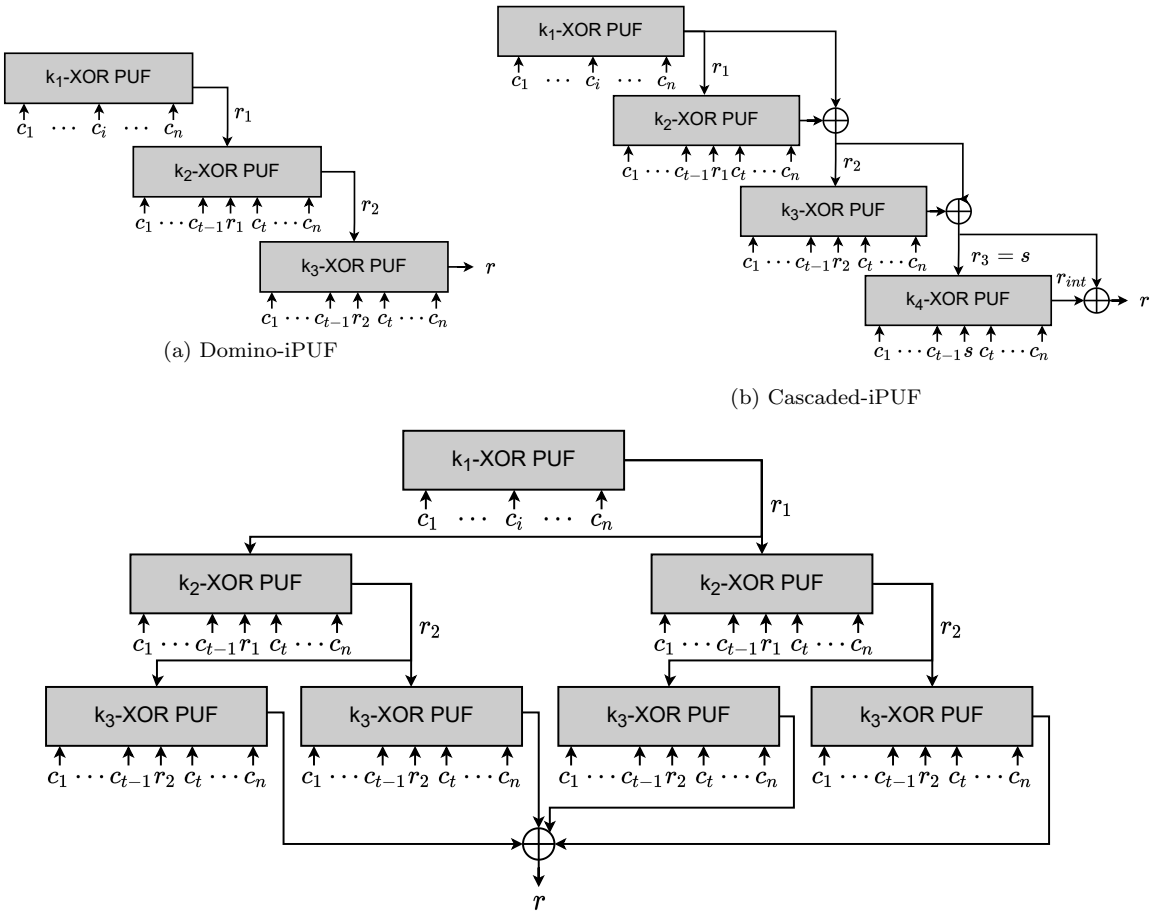
We prove the learnability of variants of iPUF proposed in [9].

3.1 PAC Learning of Domino-iPUF

A Domino-iPUF [9] consists of 3 XOR-PUFs arranged in a cascaded manner as shown in Figure 4a. The final response of the construction is the output of the lowest layer XOR-PUF.

Note that, in the last layer XOR-PUF, n out of $n+1$ bits are known. If the interpose bit and the bit position is known, the Domino-iPUF can be reduced to an $(n+1)$-bit k_3-XOR-PUF and represented using an LTF. Consequently, the final response can be calculated as follows:

$$\begin{aligned} f_{DOM} &= \prod_{j=1}^{k_3} sign(\mathbf{w}_j.\phi) = sign\Big(\bigotimes_{j=1}^{k_3} \mathbf{w}_j . \bigotimes_{j=1}^{k_3} \phi\Big) \\ &= sign(\mathbf{W}_{DOM}.\phi_{DOM}) \end{aligned} \tag{4}$$



(a) Domino-iPUF

(b) Cascaded-iPUF

(c) Tree-iPUF

Figure 4: Schematic Representation of iPUF Variants [9]

where $\mathbf{W}_{DOM} = \otimes_{j=1}^{k_3} \mathbf{w}_j$ and $\phi_{DOM} = \otimes_{j=1}^{k_3} \phi_j$ are $(n+2)^{k_3}$-dimensional vectors denoting the tensor product of the individual APUF weights and the tensor product of the parity vectors respectively. Since the interpose bit is internal to the circuit, it is chosen randomly during modelling. This in turn introduces some *noise* in the predicted responses.

To estimate the classification noise introduced by the random interpose bit, we first calculate the dependence of the final response on the interpose bit. Using theoretical bias calculation given in [8], we obtain the response bias on flipping the interpose bit as $\eta = \frac{1}{2} + 2^{k_3-1}\big(\frac{1}{2} - \frac{2}{\pi} tan^{-1}\sqrt{\frac{2t-1}{2(n+1)-2t+1}}\big)^{k_3}$. In other words, $1-\eta$ denotes the probability that the predicted response flips if the interpose bit is guessed incorrectly. Let h denote the hypothesis generated by the Perceptron algorithm and let f denote the target function (PUF). Let C_h be the set of challenges for which h outputs 1 and let C_f be the set of challenges for which f returns 1 when the interpose bit is correct. Then $C_h \triangle C_f$ denotes the set of challenges for which the hypothesis output mismatches with the PUF response, given the interpose bit guess is correct. In the PAC model, the probability that a challenge chosen with respect to input

distribution ($\mathcal{D}$) lies in $C_h \triangle C_f$ is bounded by ϵ, the maximum permissible error of hypothesis. This can be mathematically represented as $Pr[C_h \triangle C_f] \leq \epsilon$. The probability that the output of a hypothesis generated by the Perceptron algorithm (h) differs with the response obtained from the target PUF instance (r) for a challenge $\mathbf{c}$ as

$$\begin{aligned} Pr[h(\mathbf{c}) \neq r] = p &= Pr[\mathbf{c} \notin C_h \triangle C_f].Pr[r_2 \text{ impacts } r].Pr[r_2 \text{ guess is incorrect}] \\ &+ Pr[\mathbf{c} \in C_h \triangle C_f].Pr[r_2 \text{ does not impact } r] \\ &= \epsilon.\eta + ((1-\epsilon)(1-\eta).\frac{1}{2}) \\ &= \frac{(1-\eta)}{2} + \frac{\epsilon(3\eta-1)}{2} \end{aligned}$$

Thus the hypothesis output disagrees with the actual response in the following two cases – (i) when h and f disagree over an input $\mathbf{c}$ ($\mathbf{c} \in C_h \triangle C_f$) and the interpose bit has no impact on the final output and (ii) when the h and f produce the same output for a challenge $\mathbf{c}$ ($\mathbf{c} \notin C_h \triangle C_f$) and the final response gets flipped due to the incorrect interpose bit assumption.

For a ideal hypothesis, since error will occur only due to the change in response as a result of random selection of interpose bit, we obtain classification error to be $(1-\eta)/2$ by substituting $\epsilon = 0$ in Eq. 3.1. Since $\eta > 1/2$, the classification noise is less than $1/2$, which implies that the learning algorithm can still model the PUF. However, the classification error decreases the margin of the hypothesis to $\epsilon(3\eta-1)/2$. Thus, the updated mistake bound of the Perceptron algorithm is $N_{mis} = \left(\frac{2R}{\epsilon(3\eta-1)}\right)^2 = \frac{4(n+2)^{k_3}}{\epsilon^2(3\eta-1)^2}$ where $R = (n+2)^{k_3}$ is the length of the transformed input vector. Using Theorem 2.3, we obtain the sample complexity of the learning algorithm to be $\mathcal{O}\Big(\frac{2}{\epsilon(3\eta-1)}\big(log(\frac{1}{\delta}) + N_{mis}\big)\Big) = \mathcal{O}\Big(\frac{2log(1/\delta)}{\epsilon(3\eta-1)} + \frac{32d^2(n+2)^{k_3}}{\epsilon^3(3\eta-1)^3}\Big)$. Since the sample complexity is $poly(n, d, 1/\epsilon, 1/\delta)$, Domino-iPUF is PAC learnable. Note that in our learnability proof, we have assumed the interpose bit position to be fixed. However, the learnability result still holds even if the interpose bit position is set at random.

3.2 PAC Learning of XOR Cascaded-iPUF

Let us consider the example of Cascaded PUF [9] shown in Figure 4b. It consists of multiple XOR-PUFs arranged in a cascaded manner, where the interpose bit in the i^{th} layer is the XOR combination of the interpose bits in the previous $i-1$ layers. The interpose bit fed to the last layer PUF is XORed with the output of the final layer PUF to obtain the response.

Similar to Domino-iPUF, we represent the last layer XOR PUF using LTF. Let h denote the hypothesis produced by the Perceptron algorithm and f denote the target function. Here s denotes the interpose bit in the last layer that is also XORed with the last layer PUF output to generate the final response. Let $q = Pr[s = 1]$. Then, the probability that h disagrees with a CRP $(\mathbf{c}, r)$, where $\mathbf{c}$ is chosen with respect to input distribution $\mathcal{D}$ is given by

$$\begin{aligned} Pr[h(\mathbf{c}) \neq r] &= Pr[\mathbf{c} \notin C_h \triangle C_f].Pr[\text{s impacts } r] \\ &+ Pr[\mathbf{c} \in C_h \triangle C_f].Pr[\text{s does not impact } r] \\ &= Pr[\mathbf{c} \notin C_h \triangle C_f].\big(Pr[\text{s impacts } r_{in}].Pr[r = r_{in}] \\ &+ Pr[\text{s does not impact } r_{in}].Pr[r \neq r_{in}]\big) \\ &+ Pr[\mathbf{c} \in C_h \triangle C_f].\big(Pr[\text{s impacts } r_{in}].Pr[r \neq r_{in}] \\ &+ Pr[\text{s doesn't impact } r_{in}].Pr[r = r_{in}]\big) \\ &= \epsilon\big(\eta.q + (1-\eta)(1-q)\big) \\ &+ (1-\epsilon)\big(\eta(1-q) + (1-\eta)q\big) \\ &= (\eta + q - 2\eta q) + \epsilon(1 - 2\eta - 2q + 4\eta q) \end{aligned} \tag{5}$$

Since h corresponds to the hypothesis for the last layer XOR PUF including the XOR gate, $\eta = \frac{1}{2}+2^{k_4}\big(\frac{1}{2}-\frac{2}{\pi}tan^{-1}\sqrt{\frac{2t-1}{2(n+1)-2t+1}}\big)^{k_4}$ where t denotes the interpose bit position. In this case, the classification error rate for an ideal hypothesis ($\epsilon = 0$) is $q+\eta(1-2q)$. The classification noise rate is less than 0.5 if $q > 0.5$. Thus, the updated mistake bound of the Perceptron algorithm is $N_{mis} = \frac{(n+2)^{k_4}}{\epsilon^2(1-2\eta-2q+4\eta.q)^2}$ and the sample complexity is $\mathcal{O}\Big(\frac{1}{\epsilon(1-2\eta-2q+4\eta.q)}\big(log(\frac{1}{\delta})+N_{mis}\big)\Big) = \mathcal{O}\Big(\frac{log(1/\delta)}{\epsilon(1-2\eta-2q+4\eta.q)}+\frac{4d^2(n+2)^{k_4}}{\epsilon^3(1-2\eta-2q+4\eta.q)^3}\Big)$. On the other hand, if $q < 0.5$ the Perceptron algorithm learns the complement of the target PUF functionality.

3.3 PAC Learning of XOR-Domino-iPUF

Let us consider the example of XOR-Domino-iPUF that consists of k Domino-iPUFs whose outputs are combined using an XOR gate. From Eg. 3.1, we gather that Domino-iPUF is PAC learnable under LTF with noise representation class. Assuming the interpose bit is known, we can represent each Domino-iPUF output as

$$y_i = sign(\mathbf{W}_{DOM,i}.\phi_{DOM}), \qquad i \in [k]$$

Since the interpose bit is unknown and chosen at random, the outputs generated by the models have a classification error of $(1-\eta)/2$ (refer Eq. 3.1). The XOR of these outputs can be given as follows

$$\begin{aligned} f_{XDOM} &= \prod_{j=1}^{k} sign(\mathbf{W}_{DOM,j}.\phi_{DOM}) \\ &= \bigotimes_{j=1}^{k}\mathbf{W}_{DOM,j}.\bigotimes_{j=1}^{k}\phi_{DOM} \\ &= sign(\mathbf{W}_{XDOM}.\phi_{XDOM}) \end{aligned} \tag{6}$$

where $\mathbf{W}_{XDOM} = \otimes_{i=1}^{k}\mathbf{W}_{DOM,i}^{T}$ is the tensor product of the weight vectors and $\phi_{XDOM} = \otimes_{i=1}^{k}\phi_{DOM}$ is the tensor product of the parity vectors. Assuming a (k_1, k_2, k_3)-Domino-iPUF, ϕ_{XDOM} and $\mathbf{W}_{XDOM}$ are $(n+2)^{k.k_3}$-dimensional vectors. Given η to be the response bias of Domino-iPUF, the classification error of XOR-Domino-iPUF can be calculated using Piling up lemma and is given by $1-\eta' = \frac{1}{2}-2^{k-1}\eta^k$. From Eq. 6, we know that an XOR composition of k $n+1$-dimensional LTFs results in an $(n+1)^k$-dimensional LTF. Thus, we can represent an XOR-Domino-iPUF using LTF. Since the classification error is less than $1/2$, for a constant k, XOR-Domino-iPUF can be learned using the Perceptron algorithm. The mistake bound and the sample complexity of the construction can be calculated as described in [1] and in Eg. 3.1. The sample complexity of the Perceptron algorithm is $\mathcal{O}\big(\frac{2}{\epsilon(3\eta'-1)}\big(log(\frac{1}{\delta})+\frac{32d^2(n+2)^{k.k_3}}{\epsilon^2(3\eta'-1)^2}\big)\big)$ which is $poly(n, d, 1/\epsilon, 1/\delta)$. This proves that XOR-Domino-iPUF is PAC Learnable using LTF representation.

3.4 Tree-iPUF

Tree-iPUF [9] consists of XOR-PUFs arranged in a hierarchical manner (refer Figure 4c). The output of the top layer XOR-PUF is interposed in the input of both the second layer XOR-PUFs and their outputs are then interposed in the third layer XOR-PUFs. Finally, the output of the last layer XOR PUFs are combined using an XOR function to generate the PUF response. A

generic Tree-iPUF construction can consist of more than three layers as well. However, we restrict it to 3 layers for ease of exposition.

Analogous to Domino-iPUF, we adopt the LTF with noise class to represent the XOR PUFs in the last layer of the Tree-iPUF construction, assuming the interpose bit in each of the XOR-PUFs to be random. For ease of exposition, we have assumed that the number of XOR chains in all the last layer PUFs is the same (say k_3-XOR PUF). The representations for the individual XOR PUFs ($(n+2)^{k_3}$-bit LTF) can be composed into LTF (with noise) of length $(n+2)^{4k_3}$ due to the XOR operation. As shown in Figure 4c, the first two XOR PUF inputs are interposed with the same bit. Similarly, the last two XOR PUFs also share the same interpose bit. Let us consider these two bits to be set randomly. Let η_1 and η_2 be the response bias of the first two and the last two XORPUFs respectively. Then, the response bias of the final output is $\eta = \frac{1}{2} + 2^3(\eta_1 - \frac{1}{2})^2(\eta_2 - \frac{1}{2})^2$. As explained in Section 3.1, the probability that the hypothesis generated by the Perceptron algorithm disagrees with the target PUF is given by $p = \frac{(1-\eta)}{2} + \frac{\epsilon(3\eta-1)}{2}$. Therefore the classification noise imparted by the randomly chosen interpose bits is $(1-\eta)/2$. Since the classification noise is less than $1/2$, the Perceptron algorithm can PAC learn the target function. The updated margin of the hypothesis is $\epsilon(3\eta-1)/2$. Thus, the updated mistake bound of the Perceptron algorithm is $N_{mis} = \left(\frac{2R}{\epsilon(3\eta-1)}\right)^2 = \frac{4(n+2)^{4k_3}}{\epsilon^2(3\eta-1)^2}$ where $R = (n+2)^{4k_3}$ is the length of the transformed input vector. The sample complexity of the learning algorithm is $\mathcal{O}\Big(\frac{2}{\epsilon(3\eta-1)}\big(log(\frac{1}{\delta}) + N_{mis}\big)\Big) = \mathcal{O}\Big(\frac{2log(1/\delta)}{\epsilon(3\eta-1)} + \frac{32d^2(n+2)^{4k_3}}{\epsilon^3(3\eta-1)^3}\Big)$. Since the sample complexity is $poly(n, d, 1/\epsilon, 1/\delta)$, it proves that Tree-iPUF is PAC Learnable.

4 Conclusion

We proved that iPUF variants such as Domino-iPUF, XOR-Cascaded-iPUF, XOR-Domino-iPUF and Tree-iPUF are learnable in the PAC model. The sample complexity of these constructions is polynomial in terms of the challenge length, the classification noise introduced by the interpose bit and the PAC model parameters.

References

[1] Durba Chatterjee, Debdeep Mukhopadhyay, and Aritra Hazra. Interpose puf can be pac learned. *IACR Cryptol. ePrint Arch.*, 2020:471, 2020.

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

[3] 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.

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

[5] Daihyun Lim. Extracting secret keys from integrated circuits. 2005.

[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] Akhilesh Anilkumar Siddhanti, Srinivasu Bodapati, Anupam Chattopadhyay, Subhamoy Maitra, Dibyendu Roy, and Pantelimon Stănică. Analysis of the strict avalanche criterion in variants of arbiter-based physically unclonable functions. In *International Conference on Cryptology in India*, pages 556–577. Springer, 2019.

[9] Nils Wisiol, Christopher Mühl, Niklas Pirnay, Phuong Ha Nguyen, Marian Margraf, Jean-Pierre Seifert, Marten van Dijk, and Ulrich Rührmair. Splitting the interpose PUF: A novel modeling attack strategy. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):97–120, 2020.