Why Attackers Win: On the Learnability of XOR Arbiter PUFs

Fatemeh Ganji(✉), Shahin Tajik, and Jean-Pierre Seifert

Security in Telecommunications, Technische Universität Berlin
and Telekom Innovation Laboratories, Berlin, Germany
{fganji,stajik,jpseifert}@sec.t-labs.tu-berlin.de

Abstract. Aiming to find an ultimate solution to the problem of secure storage and hardware authentication, Physically Unclonable Functions (PUFs) appear to be promising primitives. While arbiter PUFs utilized in cryptographic protocols are becoming one of the most popular PUF instances, their vulnerabilities to Machine Learning (ML) attacks have been observed earlier. These attacks, as cost-effective approaches, can clone the challenge-response behavior of an arbiter PUF by collecting a subset of challenge-response pairs (CRPs). As a countermeasure against this type of attacks, PUF manufacturers shifted their focus to non-linear architectures, such as XOR arbiter PUFs with a large number of arbiter PUF chains. However, the natural question arises whether an XOR arbiter PUF with an arbitrarily large number of parallel arbiter chains can be considered secure. On the other hand, even if a mature ML approach with a significantly high accuracy is adopted, the eventual delivery of a model for an XOR arbiter PUF should be ensured. To address these issues, this paper presents a respective PAC learning framework. Regarding our framework, we are able to establish a theoretical limit on the number of arbiter chains, where an XOR arbiter PUF can be learned in polynomial time, with given levels of *accuracy* and *confidence*. In addition, we state how an XOR arbiter PUF with noisy responses can be provably PAC learned. Finally, on the basis of learning theory concepts, we conclude that no secure XOR arbiter PUF relying on current IC technologies can be manufactured.

1 Introduction

An increasing demand for secure storage of encryption mechanisms as well as hardware fingerprinting stimulates research on possible solutions. Techniques depending on storing a secret key in non-volatile memory (NVM) have been shown to be subject to physical attacks [12]. Other methods relying on the implementation of cryptographic primitives are less practical due to the constraints of the IC technology [5]. To deal with the above-mentioned issues, Physically Unclonable Functions (PUFs) have been introduced [8,20]. From a general point of view, the security-related functionality of PUFs, more specifically their challenge-response behavior, is offered by the manufacturing variations of an IC. One of the most celebrated types of PUF instances are arbiter PUFs, which

M. Conti et al. (Eds.): TRUST 2015, LNCS 9229, pp. 22–39, 2015.
DOI: 10.1007/978-3-319-22846-4_2

are widely utilized in several different cryptographic protocols [5,13,17]. The challenge-response behavior of an arbiter PUF is characterized by slightly different propagation delays of identical paths, caused by chip imperfections. These slight differences are further exploited to generate unique responses.

While authentication and fingerprinting methods enjoying this privilege have been emerging, it has been demonstrated that arbiter PUFs are vulnerable to different types of attacks. Different ML techniques contribute to the success of non-invasive ML attacks against arbiter PUFs [13]. Aiming at mathematically *cloning* an arbiter PUF, the attacker collects a set of challenge-response pairs (CRPs), and *attempts* to provide a model that can *approximately* predict the response of the PUF to an arbitrarily chosen challenge. Most of the ML attacks benefit from the linear additive model of an arbiter PUF. This forces a migration to modified structures of arbiter PUFs, in which non-linear effects are added to the PUF in order to impair the effectiveness of ML attacks. To this end, XORing the responses of multiple arbiter PUFs has been demonstrated as a promising solution [28].

However, it has been shown that more advanced ML techniques can still break the security of an XOR arbiter PUF (briefly called XOR PUF in this paper) with a limited number of arbiter chains (here called chains) [23]. Going beyond this limited number is suggested as a countermeasure by PUF manufacturers, although they have encountered serious problems, namely the increasing number of noisy responses as well as optimization of the silicon area required on the respective chip [21]. Even in this case, physical side-channel attacks, such as photonic emission analysis, can physically characterize XOR PUFs regardless of the number of XORs [29]. In another attempt a combination of ML attacks with non-invasive side channel attacks (e.g., power and timing) is suggested to model XOR PUFs, with the number of chains exceeding the previously established limit [24].

The latter attack is cost-effective due to its non-invasive nature, and therefore, it might be preferred to the semi-invasive one in practice. However, in contrast to pure ML techniques (i.e., without any side channel information), using side channel information in combination with ML techniques requires physical access to the device and reconfiguration of the circuits on the chip, which are not always feasible in a real scenario [24]. Therefore, it is still tempting to develop new pure ML techniques to break the security of XOR PUFs, with an arbitrary number of chains. Nevertheless, it is still unclear how many chains should be XORed to ensure the security of arbiter PUFs against ML attacks. Moreover, when applying current ML attacks, the maximum number of CRPs required for modeling an XOR PUF, with given levels of *accuracy* and final model delivery *confidence*, is not known today.

Only recently, it has been shown how a single chain arbiter PUF under the Deterministic Finite Automata (DFA) representation can be learned for given levels of accuracy and confidence [7]. It is further proved that the run time of their proposed algorithm is polynomial in the size of the DFA. We claim that for the XOR PUFs, a more compact representation can be adopted to improve the time complexity of this attack. Furthermore, to deal with noisy responses of an XOR PUF more efficiently, in contrast to their method, an approach not relying on majority voting can be applied.

We present a new framework to prove that XOR PUFs can be learned in polynomial time, for given levels of accuracy and confidence. The main contributions of our framework are summarized as follows:

Finding a theoretical limit for ML techniques to learn XOR PUFs in polynomial time. Under a well-known representation of an XOR PUF, we provide a theoretical limit as a function of the number of arbiter PUF stages and the number of chains, where an XOR PUF can be provably learned in polynomial time.

Learning of an XOR PUF for given levels of accuracy and confidence. With regard to the proposed limit, we present an algorithm, which learns the challenge-response behavior of an XOR PUF, for given levels of accuracy and confidence. The run time of this algorithm is polynomial in the number of the arbiter PUF stages, the number of chains, as well as the levels of accuracy and confidence. Moreover, our approach requires no side channel information.

Modeling the XOR PUF even if the responses are noisy. A celebrated model of noise fitting the purpose of our ML framework is applied to prove that even in the presence of noise, the run time of our algorithm is still polynomial in the number of the arbiter PUF stages, the number of chains, levels of accuracy and confidence, and the noise rate. Finally, through a comprehensive discussion, we will explain why secure XOR PUFs cannot be manufactured on chips based on current technologies.

2 Notation and Preliminaries

This section focuses on the background information and notations required to understand the general concept of arbiter PUFs, XOR PUFs, fundamentals of LTFs, the perceptron algorithm, the PAC model, and finally PAC learning with the Perceptron algorithm.

2.1 Arbiter and XOR PUFs

PUFs are most often related to the intrinsic silicon properties of a chip. They are physical input to output mappings, which generate a *response* for a given *challenge*. Let $\mathcal{C} = \{0,1\}^n$ and $\mathcal{Y} = \{0,1\}$ be the set of challenges and the set of responses, respectively. A PUF can be described by the function $f_{\mathrm{PUF}} : \mathcal{C} \rightarrow \mathcal{Y}$ where $f_{\mathrm{PUF}}(c) = y$.

PUFs are *evaluable*, which means that for a given PUF, f_{PUF} can be evaluated in polynomial time. Given a set of PUF instantiations, each PUF is *unique* and different from other PUFs with regards to its response set $\mathcal{Y}$. A response $y = f_{\mathrm{PUF}}(c)$ is *reproducible* in a sense that different evaluations of the same challenge yield "close" responses with respect to the considered distance metric.

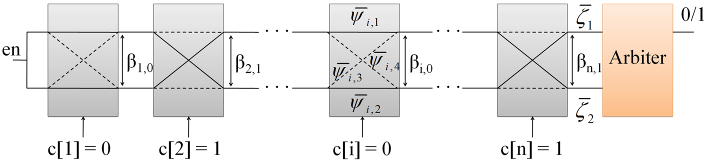


Fig. 1. Schematic of an arbiter PUF with n multiplexer stages and an arbiter at the end of the chain. Each multiplexer stage consists of four different delays. Based on the applied challenge, when the enable signal (denoted by "en") is fed, either the direct paths or the crossed paths are utilized for the signal propagation. Upon arrival of the first signal, the arbiter generates a binary response.

As the name implies, PUFs are *unclonable*, i.e., it is nearly impossible to construct another physical mapping (device) g_{PUF}, where $g_{\text{PUF}} \neq f_{\text{PUF}}$, but g_{PUF} and f_{PUF} have a similar challenge response behavior. Moreover, PUFs are *unpredictable*, which means that despite knowing a set $U = \{(c_i, y_i) \mid y_i = f_{\text{PUF}}(c_i)\}$, it is practically impossible to predict $y_r = f_{\text{PUF}}(c_r)$, where c_r is a random challenge with $(c_r, \cdot) \notin U$. Finally, PUFs are *one-way*, i.e., for a given $y = f_{\text{PUF}}(c)$, the probability that a probabilistic polynomial time algorithm or a physical procedure $\mathcal{A}$ can output c is negligible, where c is drawn from a uniform distribution on $\{0,1\}^n$ [25].

Utilizing the timing differences of symmetrically designed electrical paths on a chip is the core idea of arbiter PUFs. The chain of an arbiter PUF consists of n connected switches, or so called *stages*, and an arbiter at the end, see Fig. 1. A challenge is an n-bit string $c = c[1] \cdots c[n]$, where the i^{th} bit is fed into the i^{th} stage. There are four different paths in each stage. If $c[i] = 1$, the signal propagates through the crossed paths, otherwise the direct paths are utilized, see Fig. 1. Enabling the inputs of the first stage leads to the propagation of two electrical signals on two symmetrically designed paths terminated by the arbiter. Due to the imperfections on the chip the two signals arrive at the end of the chain at different times. With regard to the arrival time of the signals, the arbiter generates a binary response.

We define a random variable $\varPsi_i$ related to the delay within the i^{th} stage, which follows a Gaussian distribution with the mean μ_i and the deviation σ_i [7,22]. The realizations of the random variable $\varPsi_i$ are certain $\overline{\psi}_{i,1}$, $\overline{\psi}_{i,2}$, $\overline{\psi}_{i,3}$, and $\overline{\psi}_{i,4}$. $\overline{\psi}_{i,1}$ and $\overline{\psi}_{i,2}$ are the delays of the upper and lower direct paths, whereas $\overline{\psi}_{i,3}$ and $\overline{\psi}_{i,4}$ are the delays of the upper and lower crossed paths, respectively, see Fig. 1. The delay differences between the upper and lower outputs of the i^{th} stage are denoted by $\overline{\beta}_{i,0} = \overline{\psi}_{i,1} - \overline{\psi}_{i,2}$ and $\overline{\beta}_{i,1} = \overline{\psi}_{i,4} - \overline{\psi}_{i,3}$, for direct paths and crossed paths, respectively.

Let $\mathcal{Z}$ be a random variable which corresponds to the total delay between the enable point and the outputs of the n^{th} stage of the arbiter PUF. With regard to the linear additive model of an arbiter PUF, we have $\mathcal{Z} = \sum_{l=1}^{n} \varPsi_l$ [7,13]. $\overline{\zeta}_1$ and $\overline{\zeta}_2$ are the realizations of $\mathcal{Z}$ at the upper and lower output, respectively, see Fig. 1. Let $\kappa > 0$ denote the precision of the arbiter. By comparing $\overline{\zeta}_1$ and $\overline{\zeta}_2$,

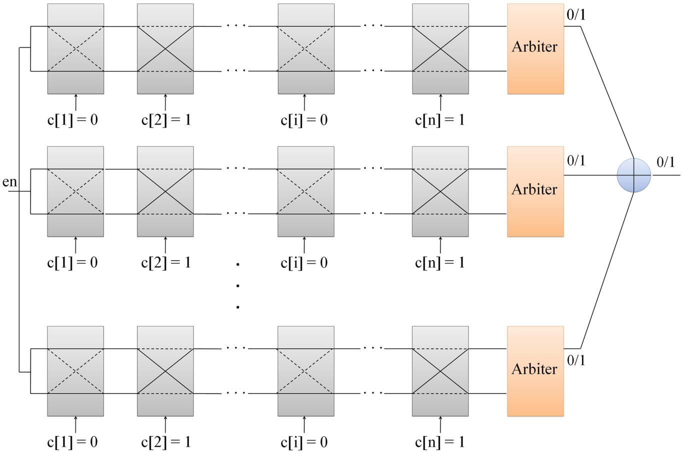


Fig. 2. Schematic of an XOR PUF. It consists of k chains of n-bit arbiter PUFs. The responses of all arbiters are XORed together to generate the final binary response.

the arbiter makes a decision whether the output is either "1" or "0". More formally, we assume that the output of the PUF is "1" if $\overline{\Delta} = \overline{\zeta}_1 - \overline{\zeta}_2 > \kappa$, whereas it is "0" if $\overline{\Delta} < -\kappa$. If $|\overline{\Delta}| \leq \kappa$, the arbiter is assumed to be in a metastable condition.

Following the procedure introduced in [7], in each stage, e.g., i^{th} stage, $\overline{\psi}_{i,j}$ can be mapped into an integer value $\psi_{i,j}$ $(1 \leq j \leq 4)$. It is known that $\overline{\psi}_{i,j} \in [\mu_i - 3\sigma_i, \mu_i + 3\sigma_i]$ with probability 99.7 %. Now we define the mapping $f_{int} : \mathbb{R} \mapsto \mathbb{Z}$ so that for all $\overline{\psi}_{i,j} \in [\mu_i - 3\sigma_i, \mu_i + 3\sigma_i]$, we have $\psi_{i,j} = f_{int}(\overline{\psi}_{i,j}) = \left\lceil (\overline{\psi}_{i,j} - \mu_i + 3\sigma_i)/\kappa \right\rceil$. Without loss of generality, we assume that $\mu_1 = \cdots = \mu_n$ and $\sigma_1 = \cdots = \sigma_n$. Hence, by performing the mapping f_{int} the maximum and the minimum of the real valued delays $\overline{\psi}_{i,j}$ $(1 \leq i \leq n$ and $1 \leq j \leq 4)$ are mapped into $m = \left\lceil \frac{6\sigma}{\kappa} \right\rceil$ and 0, respectively (for more details see [7]). Furthermore, similarly, $\overline{\Delta}$ can be mapped with a high probability to an integer value Δ lying within a finite interval. In this case, the response of the arbiter is "1" if $\Delta > 0$, whereas it is "0" if $\Delta < 0$. The arbiter is in the metastable condition, if $\Delta = 0$.

To improve the security of arbiter PUFs against machine learning attacks, a modified construction called XOR PUF was suggested by [28]. An XOR PUF consists of k different chains, all with the same number of stages n. The responses of all arbiter PUFs are XORed together to generate the final response, see Fig. 2. The response of the XOR PUF can be defined as $f_{XOR}(c) = \bigoplus_{j=1}^{k} f_{j^{\text{th}}\text{arbiter PUF}}(c)$.

2.2 Linear Threshold Functions

We begin with the definition of a Perceptron (i.e., single-layer Perceptron), where P_n denotes an n-input and single output Perceptron. P_n is represented by the function $\Omega \rightarrow H$, where the vector $\boldsymbol{\omega} = (\boldsymbol{\mathcal{A}}[1], \boldsymbol{\mathcal{A}}[2], \cdots, \boldsymbol{\mathcal{A}}[n], \theta)$ denotes a state, and the set $\Omega \subset \mathbb{R}^{n+1}$ is the set of states. The function $h_\omega \in H$ with $h_\omega : \mathbb{R}^n \mapsto \{0, 1\}$ is defined as follows:

$$h_\omega = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} \boldsymbol{\mathcal{A}}[i]\boldsymbol{\Phi}[i] - \theta \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The sets of positive and negative examples of h_ω are half-spaces S^1 and S^0, where $S^1 = \{\boldsymbol{\Phi} \in \mathbb{R}^n | \sum_{i=1}^{n} \boldsymbol{\mathcal{A}}[i]\boldsymbol{\Phi}[i] \geq \theta\}$ and $S^0 = \{\boldsymbol{\Phi} \in \mathbb{R}^n | \sum_{i=1}^{n} \boldsymbol{\mathcal{A}}[i]\boldsymbol{\Phi}[i] < \theta\}$. Alternatively, by applying the mapping $f_{\text{map}} : \{0,1\} \mapsto \{1, -1\}$, so that $f_{\text{map}}(0) = 1$ and $f_{\text{map}}(1) = -1$, we have:

$$h_\omega = sgn(\boldsymbol{\Phi} \cdot \boldsymbol{\mathcal{A}} - \theta), \tag{2}$$

where the inner product of the vector $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\Phi}$ is denoted by $\boldsymbol{\Phi} \cdot \boldsymbol{\mathcal{A}}$. Equation (2) denotes a linear threshold function (LTF), whose decision regions are S^1 and S^0 bounded by the hyperplane $\mathcal{P} : \boldsymbol{\Phi} \cdot \boldsymbol{\mathcal{A}} = \theta$ (for further details see [2]).

2.3 Perceptron Algorithm

The Perceptron algorithm is an *online* algorithm invented to learn LTFs efficiently. By online we mean that providing the learner (i.e., learning algorithm) with each example, e.g., $\boldsymbol{\Phi}_i$, it attempts to predict the response to that example. Afterwards, the actual response (i.e., the label, for instance $\mathfrak{p}(\boldsymbol{\Phi}_i)$) is presented to the learner, and then it can improve its hypothesis by means of this information. The learning process continues until all the examples are provided to the learner [14].

Let the input of the Perceptron algorithm be a sequence of r labeled examples $((\boldsymbol{\Phi}_1, \mathfrak{p}(\boldsymbol{\Phi}_1), \cdots, (\boldsymbol{\Phi}_r, \mathfrak{p}(\boldsymbol{\Phi}_r))$. The output of the algorithm is the vector $\boldsymbol{\mathcal{A}}$ classifying the examples. Executing the Perceptron algorithm, it initially begins with $\boldsymbol{\omega_0} = (\boldsymbol{\mathcal{A}}_0[1], \boldsymbol{\mathcal{A}}_0[2], \cdots, \boldsymbol{\mathcal{A}}_0[n], \theta) = (0, \cdots, 0)$. When receiving each example (e.g., $\boldsymbol{\Phi}_j$), the algorithm examines whether $\boldsymbol{\mathcal{A}}_j[i] \cdot \boldsymbol{\Phi}_j[i] \geq \theta_j$ and compares its prediction with the received label. If the label and the prediction of the algorithm differ, ω_j is updated as follows:

$$\omega_{j+1}[k] = \begin{cases} \boldsymbol{\mathcal{A}}_j[k] - \mathfrak{p}(\boldsymbol{\Phi}_j) \cdot \boldsymbol{\Phi}_j[k] & 1 \leq k \leq n \\ \theta_j - \mathfrak{p}(\boldsymbol{\Phi}_j) & k = n + 1. \end{cases}$$

Note that if the prediction and the label of an example agree, no update is performed [26].

Quantifying the performance of an on-line algorithm, the prediction error (i.e., number of mistakes) of the algorithm is taken into account. In this way, the upper bound of the mistakes is defined as a measure of the performance. The Perceptron convergence theorem gives an upper bound of the error that can occur while executing the Perceptron algorithm [6]:

Convergence Theorem of the Perceptron Algorithm: *Considerr labeled examples which are fed into the Perceptron algorithm, and* $\|\mathbf{\Phi}_i\| \leq R$ $(\|\cdot\|$ *denotes the Euclidean length). Let* $\mathbf{u}$ *be the solution vector with*$\|\mathbf{u}\| = 1$ *whose error is denoted by*ε $(\varepsilon > 0)$*. The deviation of each example is defined as*$d_i = \max\{0, \varepsilon - \mathfrak{p}(\mathbf{\Phi}_i)(\boldsymbol{u} \cdot \mathbf{\Phi}_i)\}$*, and* $D = \sqrt{\sum_{i=1}^{r} d_i^2}$*. The upper bound of the mistakes of the Perceptron algorithm is*

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

For the proof, the reader is referred to [6]. Let the parameter σ be the minimum distance of any example from $\mathcal{P}$, i.e.,

$$\sigma = \min_{\mathbf{\Phi} \in \mathit{\Phi}} \frac{\|\mathbf{\Phi} \cdot \boldsymbol{\mathcal{A}}\|}{\|\boldsymbol{\mathcal{A}}\|}, \tag{3}$$

where $\mathit{\Phi}$ is the set of all $\mathbf{\Phi}$'s. The order of $1/\sigma$ determines whether the data is linearly separable. It has been demonstrated that when $1/\sigma$ is exponential in n, the data is not linearly separable, and consequently, the Perceptron algorithm cannot classify the data [4,26]. On the other hand, if $1/\sigma$ is polynomial in n, the Perceptron algorithm can be applied.

2.4 PAC Model

As the name implies, the concept of PAC (Probably Approximately Correct) model aims at learning an unknown target (i.e., a concept class $\mathfrak{C}_n$) under the following circumstances: (a) after the learning phase, the output of the algorithm is a hypothesis approximating $\mathfrak{C}_n$, and (b) with a high probability the learner can deliver a good hypothesis.

To formalize the above mentioned definition, let $\mathfrak{C}_n$ be defined over the instance space $X_n = \{0,1\}^n$. Furthermore, $X = \cup_{n \geq 1} X_n$ and $\mathfrak{C} = \cup_{n \geq 1} \mathfrak{C}_n$. We have also $\mathfrak{H}_n$ as the *hypothesis space*, and $\mathfrak{H}$ defined in a similar fashion. The learner is provided with a finite number of examples drawn randomly with respect to the probability distribution D. For the target concept $c \in \mathfrak{C}$, the error of the hypothesis is $\mathit{error}(h) := \mathit{\Sigma}_{x \in h \triangle c} D(x)$, where $\triangle$ is the symmetric difference. Now we can define PAC learnability as followings.

Let $p(\cdot,\cdot,\cdot)$ be a polynomial, ε and δ be arbitrary values such that $0 < \varepsilon, \delta < 1$, and a distribution D on the instance space X_n (arbitrary distribution). When a PAC learning algorithm $\mathcal{L}$ is fed by $p(n,1/\varepsilon,1/\delta)$ independent examples drawn randomly with respect to D, then with probability at least $1 - \delta$ the output of $\mathcal{L}$ is a hypothesis $h \in \mathfrak{H}_n$ such that $\mathit{error}(h) \leq \varepsilon$. The sample complexity of $\mathcal{L}$ is the smallest polynomial p. With regard to the relation between $\mathfrak{C}$ and $\mathfrak{H}$, we define that $\mathfrak{C}$ is *properly* PAC learnable, when $\mathfrak{C} = \mathfrak{H}$. Otherwise, if $\mathfrak{H}$ can be evaluated on given instances in polynomial time, and $\mathfrak{C}$ is PAC learnable by $\mathfrak{H}$, $\mathfrak{C}$ is called *PAC learnable.*

2.5 PAC Learning of LTFs with Perceptron Algorithm

Several studies have focused on the PAC learning of an unknown LTF from labeled examples by applying the Perceptron algorithm (for an exhaustive survey see [26]). Here we briefly describe how the Perceptron algorithm, as an online algorithm, can be converted to a PAC learning algorithm, following the conversion procedure defined in [26].

The learner has access to an Oracle EX, providing labelled examples. By calling EX successively, a sequence of labeled examples is obtained and fed into the online algorithm. Hypotheses generated by the algorithm are further stored. At the second stage, the algorithm again calls EX to receive a new sequence of labeled examples. This new sequence is used to calculate the error rate of the hypotheses stored beforehand. The output of the procedure is a hypothesis with the lowest error rate. Let ε and δ be the accuracy and the confidence levels of the obtained PAC learning algorithm. Suppose that N_{mis} is the upper bound of the mistakes made by the original online algorithm for the concept class $\mathfrak{C}$. The following theorem is proved by Littlestone [15].

Theorem 1. *Suppose that the online algorithm L_{on} improves its hypothesis, only when its prediction and the received label of the example do not agree. The total number of calls that the obtained PAC algorithm $\mathcal{L}$ makes to EX is $O\left(1/\varepsilon(\log 1/\delta + N_{mis})\right)$.*

From the convergence theorem of the Perceptron algorithm and Theorem 1, it is straightforward to prove the following corollary [26]:

Corollary 1. *Let the concept class $\mathfrak{C}_n$ over the instance space $X_n = \{0,1\}^n$ be the class of linear threshold functions such that the weights $\mathcal{A}_i \in \mathbb{Z}$, and $\sum_{i=1}^{n} |\mathcal{A}[i]| \leq \mathfrak{A}$, where $\mathfrak{A} \in \mathbb{Z}$ is the maximum sum over the weights. Then the Perceptron algorithm can be converted to a PAC learning algorithm running in time $p(n, \mathfrak{A}, 1/\varepsilon, 1/\delta)$.*

3 PAC Learning of XOR PUFs

In this section we first present how and why an XOR PUF can be PAC learned by the Perceptron algorithm. Furthermore, we provide the theoretical limit for the learnability of XOR PUFs in polynomial time. Finally, the theoretical results will be verified against experimental results from existing literature.

3.1 LTF-Based Representation of XOR PUFs

Here we briefly describe the LTF-based representation of an XOR PUF, which is widely adopted [9,19,23]. Consider the delay vector $\mathcal{A}^T$ defined as follows:

$$\mathcal{A}^T = (\alpha_1, \alpha_2, \ldots, \alpha_{n+1}) \text{ with } \begin{cases} \alpha_1 = \frac{\beta_{i,0}-\beta_{i,1}}{2} \\ \alpha_i = \frac{\beta_{i-1,0}+\beta_{i-1,1}+\beta_{i,0}-\beta_{i,1}}{2}, 2 \leq i \leq n \\ \alpha_{n+1} = \frac{\beta_{n,0}+\beta_{n,1}}{2} \end{cases} \tag{4}$$

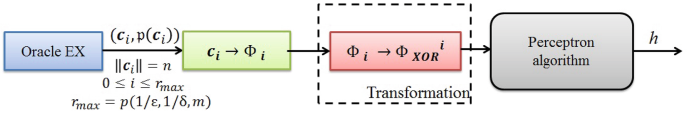

Oracle EX
$(\boldsymbol{c}_i, \mathrm{p}(\boldsymbol{c}_i))$
$\|\boldsymbol{c}_i\| = n$
$0 \le i \le r_{max}$
$r_{max} = p(1/\varepsilon, 1/\delta, m)$
$\boldsymbol{c}_i \to \Phi_i$
$\Phi_i \to \Phi_{XOR}{}^i$
Transformation
Perceptron algorithm
h

Fig. 3. Block diagram of the PAC learning framework. By calling the Oracle *EX* at most r_{max} times, a sequence of examples is collected. $\boldsymbol{\Phi}_i$ is fed into the third block corresponding to our problem transformation. The output of the third block is fed into the Perceptron algorithm.

where the integer valued $\beta_{i,j}$ ($1 \le i \le n$ and $0 \le j \le 1$, as shown in Fig. 1) are the delay differences at the output of the i^{th} stage. Wrt. the discretization process described in Sect. 2.1, it is straightforward to show that $\beta_{i,j}$ lies within the interval $[-m, m]$, hence, α_i ($1 \le i \le n+1$) lies within the interval $[-2m, 2m]$.

Consider a challenge string represented by a vector $\mathbf{c} = (c[1], \cdots, c[n])$. $\boldsymbol{\Phi} = (\boldsymbol{\Phi}[1], \cdots, \boldsymbol{\Phi}[n], 1)$ is the encoded challenge vector, where $\boldsymbol{\Phi}[i] = \prod_{j=i}^{n}(1 - 2c[j])$. We defined Δ as the delay difference at the outputs of the last stage. According to the linear additive model of the arbiter PUF, we have $\Delta = \boldsymbol{\mathcal{A}}^T \cdot \boldsymbol{\Phi}$, cf. [13,23]. Now let $f_{\text{map}} : \mathcal{Y} \mapsto \{1, -1\}$, so that $f_{\text{map}}(0) = 1$ and $f_{\text{map}}(1) = -1$. The output of the arbiter can be defined as

$$f_{PUF} = sgn(\Delta) = sgn(\boldsymbol{\mathcal{A}}^T \cdot \boldsymbol{\Phi}). \tag{5}$$

From Eq. (5), it is obvious that an arbiter PUF can be represented by an $(n+1)$-dimensional LTF. In a similar fashion, an XOR PUF can also be represented by an LTF, when the final response ($\mathcal{Y}_{XOR}$) is mapped to $\{1, -1\}$, cf. [23]:

$$f_{XOR} = \prod_{j=1}^{k} sgn(\boldsymbol{\mathcal{A}}^T \cdot \boldsymbol{\Phi}) = sgn\left(\bigotimes_{j=1}^{k} \boldsymbol{\mathcal{A}}^T \cdot \bigotimes_{j=1}^{k} \boldsymbol{\Phi}_j\right) = sgn(\boldsymbol{\mathcal{A}}_{XOR}^T \cdot \boldsymbol{\Phi}_{XOR}) \tag{6}$$

where $\boldsymbol{\mathcal{A}}_{XOR} = \otimes_{j=1}^{k} \boldsymbol{\mathcal{A}}_j^T$ is the tensor product of the vectors $\boldsymbol{\mathcal{A}}_j^T$, and similarly $\boldsymbol{\Phi}_{XOR} = \otimes_{j=1}^{k} \boldsymbol{\Phi}_j$.

3.2 PAC Learning of XOR PUFs with Perceptron

We claim that by adopting a simple transformation the Perceptron algorithm can be applied, particularly in our case. Comparing Eqs. (5) and (6), it can be seen that the $(n+1)^k$-dimensional vectors $\mathcal{A}_{XOR}$ and $\boldsymbol{\Phi}_{XOR}$ are substituted for $(n+1)$-dimensional vectors $\boldsymbol{\mathcal{A}}^T$ and $\boldsymbol{\Phi}$. In other words, XOR-ing k $(n+1)$-dimensional LTFs results in an $O((n+1)^k)$-dimensional LTF. Therefore, if we transform the problem of learning the XOR of k $(n+1)$-dimensional LTFs into an $O((n+1)^k)$ dimensional space, this problem can be solved by applying the Perceptron algorithm. In order to support our claim, we begin with the following theorem stating that examples are linearly separable in $\mathbb{R}^{n+1}$.

Theorem 2. *For an XOR PUF, with fixed k, represented by an LTF in an $O\left((n+1)^k\right)$ dimensional space, $1/\sigma$ is polynomial in n.*

Proof. As described in Sect. 2, $\boldsymbol{\mathcal{A}}_{XOR}$ contains integer elements lying within the limited interval $[-2m,2m]$. Furthermore, elements of $\boldsymbol{\Phi}_{XOR}$ are in $\{-1,1\}$. Now we have

$$\sigma = \min_{\boldsymbol{\Phi}_{XOR} \in \varPhi_{XOR}} \frac{\|\boldsymbol{\Phi}_{XOR} \cdot \boldsymbol{\mathcal{A}}_{XOR}\|}{\|\boldsymbol{\mathcal{A}}_{XOR}\|} = \min_{\boldsymbol{\Phi}_{XOR} \in \varPhi_{XOR}} \frac{\sqrt{\sum_{i=1}^{(n+1)^k} \alpha_i^2}}{\sqrt{(n+1)^k}}, \tag{7}$$

where $\varPhi_{XOR}$ is the set of all $\boldsymbol{\Phi}_{XOR}$'s. Due to a non-trivial challenge-response behavior of the given PUF, at least one of the elements of $\boldsymbol{\mathcal{A}}_{XOR}$ must be equal to ±1. Therefore, $\min \left(\sum_{i=1}^{(n+1)^k} \alpha_i^2\right)^{1/2} = 1$, and $1/\sigma = (n+1)^{k/2}$. ■

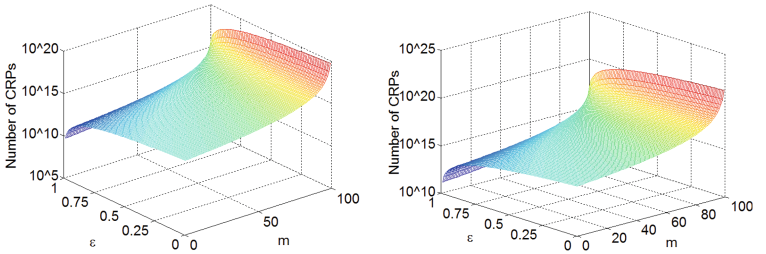
Figure 3 illustrates how the perceptron algorithm is applied in our PAC learning framework. The learner has access to an Oracle EX, which is related to the XOR PUF as follows: $EX := f_{XOR}$. At the first stage, the Oracle EX is called successively to collect CRPs. The maximum number of calls is denoted by r_{max}. For each CRP (e.g., $(\mathbf{c}_i,\mathfrak{p}(\mathbf{c}_i))$) a vector $\boldsymbol{\Phi}_i$ is generated. Afterwards, $\boldsymbol{\Phi}_i$ is transformed to $\boldsymbol{\Phi}^i_{XOR}$, which is in an $O\left((n+1)^k\right)$ dimensional space. The Perceptron algorithm predicts the response to $\boldsymbol{\Phi}^i_{XOR}$, and if its prediction and $\mathfrak{p}(\mathbf{c}_i)$ disagrees, its hypothesis will be updated.

Now we will elaborate on the upper mistake bound in our framework. Following the convergence theorem of the Perceptron algorithm and Theorem 2, when $\|\boldsymbol{\Phi}_{XOR}\| \leq (n+1)^{k/2}$, we have $N_{mis} = (n+1)^k/\varepsilon^2$. As an immediate corollary, we have:

Corollary 2. *Let k be constant and consider the class of XOR PUFs over the instance space $X_n = \{0,1\}^{(n+1)^k}$ which is that class of linear threshold functions such that $\alpha_i \in \mathbb{Z}$, and $\sum_{i=1}^{(n+1)^k} |\alpha_i| \leq 2m(n+1)^k$. Then the Perceptron-based algorithm running in time $p((n+1)^k, 4m^2, 1/\varepsilon, 1/\delta)$ can PAC learn an XOR PUF by calling EX at most $O\left(\log(1/\delta)/\varepsilon + 4m^2(n+1)^k/\varepsilon^3\right)$ times.*

There are some key implications from this corollary. First, with regard to the PAC model, the hypothesis delivered by the algorithm must be evaluable in polynomial time. The point here is that the term $(n+1)^k$, related to the Vapnik-Chervonenkis dimension of the representation [3], may grow significantly if k is not a constant. In this case, our algorithm cannot find a hypothesis in polynomial time, since the number of examples is super-polynomial in n. However, in Sect. 5 we will see that in practice, k cannot exceed the bound $\lceil \ln n \rceil$, and therefore, XOR PUFs realized in practice are PAC learnable, as k must be reasonable small and can be seen as constant when compared with n.

Second, it is tempting that an increase in m can help to ensure the security of an XOR arbiter PUF. The upper bound for the number of CRPs calculated according to the Corollary 2, for $\delta = 0.0001$, $k = 5$ for $n = 64$ and $n = 128$, is depicted in Fig. 4. This figure can provide a better understanding of the impact of



(a) $n = 64$, $k = 5$ (b) $n = 128$, $k = 5$

Fig. 4. Upper bound of the number of CRPs. The x-axis indicates m, whereas the y-axis shows ϵ. The z-axis corresponds to the upper bound of the number of CRPs.

m on the learnability of an XOR arbiter PUF. A marginal increase in m cannot dramatically increase the number of CRPs required for modeling an XOR arbiter PUF. Although an arbitrarily large m can be suggested to ensure the security of an XOR arbiter PUF, as stated in [7], m is restricted by technological limits (i.e., the yield and many other factors), and thus cannot be arbitrarily large. In Sect. 5 the impact of an increase in m on the learnability will be further discussed.

3.3 Validation of the Theoretical Results

We compare our theoretical findings with several experimental results reported in [16,23]. As reported in [16], until now no *effective* pure ML attack has been launched on XOR PUFs with $k \geq 5$ ($n = 64$). Pure ML attacks proposed in the literature are conducted on XOR PUFs with the maximum n being equal to 128 [23,24]. Taking into account the long run time of pure ML algorithms, even on the powerful machines employed by [23], XOR PUFs with $n = 256$ and $n = 512$ have been targeted only by combined modeling attacks [24]. Therefore, unfortunately, in the literature no practical limit for pure ML techniques has been reported for XOR PUFs with $n \geq 128$.

As an attempt to compare our theoretical results to what has been observed in practice, we focus on the results reported in [23]. Note that although the algorithm applied in [23] (Logistic Regression) differs from our algorithm, we can compare the number of CRPs required by them to learn an XOR PUF with a given accuracy. The argument supporting this claim is that the hypothesis class of LR can be "discretized" so that it becomes finite [27]. Furthermore, due to the fact that the delay values can be mapped to a finite interval of integer values, the loss function of LR is also bounded. Therefore, LR can be converted to a PAC learning algorithm, and the maximum number of EX calls made by the obtained algorithm is polynomial in n, k, $1/\epsilon$ and $1/\delta$. Moreover, note that the theoretical limit of learning an XOR PUF in polynomial time is established by the Vapnik-Chervonenkis dimension of the LTF representation of an XOR PUF, as used in [23] as well. These enable us to compare their experimental results with our findings.

The authors of [23] have attempted to model 64-bit and 128-bit XOR PUFs with up to 6 and 5 chains, respectively. Their results demonstrate that the proposed model can predict responses to a set of arbitrary chosen challenges with 99 % accuracy. However, the number of CRPs required for modeling a 64-bit XOR PUF with $k = 5$ and $k = 6$ is increased drastically, comparing to those with $k \leq 4$. In this regard, the number of CRPs collected to predict the response is increased from 12000 to 80000 and 200000 for $k = 5$ and $k = 6$, respectively ($\epsilon = 0.01$). As a result, the time spent to build a model is increased from a few minutes to several hours, which shows an exponential growth. For a 128-bit XOR PUF with $k = 5$, 500000 CRPs are required to model the XOR PUF, with 99 % accuracy, while for a PUF with $k = 4$, this number is only 24000. Consequently, the learning time is again increased exponentially.

In the above-mentioned cases, an exponential growth in the number of CRPs and the learning time can be clearly observed, when k exceeds 4. This matches the theoretical limit proposed in Sect. 3.2.

4 PAC Learning of Noisy XOR PUFs

In the previous section, we have explained how the Perceptron algorithm can be applied to PAC learn an XOR PUF. The natural and important question would be whether the proposed framework is applicable in the case of noisy responses. The term *noisy response* here refers to the response of the XOR PUF to a challenge under either the metastable condition or the impact of environment noise. Although it has been accepted that metastablity of an XOR PUF must be solved by the PUF manufacturer, we consider this particular case for completeness. From the point of view of PAC learning, this condition results in incorrect labels generated by the Oracle EX. We aim to state that an XOR PUF can be PAC learned by applying the Perceptron algorithm, even if noisy responses are included in the collected set of CRPs.

Several versions of the Perceptron algorithm, which can tolerate noise, i.e., incorrect labels of examples, have been developed (for a comprehensive survey see [11]). Here we follow the work by [1] to demonstrate that the original Perceptron algorithm can be further applied in the case of noisy responses. In this case, the number of CRPs required to be collected is polynomial in the number of noisy responses.

At the first stage, we define a simple but effective model of noisy Oracle EX_η [1]. In our model, the examples are drawn with respect to the relevant distribution D, and the label of each example is chosen in an independent random fashion. More specifically, after drawing an example, an unfair coin (head with probability $1-\eta$) is flipped. If the outcome is head, the correct label is provided, otherwise the label is incorrect. It is clear that $\eta < 1/2$, since $\eta = 1/2$ means that no relevant information is provided by EX_η, and the case of $\eta > 1/2$ is irrelevant. We assume that an upper bound on η, denoted by η_b, is known. Even if this assumption may not be the case in practice, following the procedure defined in [1], η_b can be estimated. It is shown that the sample size is increased very slightly in the case of unknown η_b (for further information and the proof see [1]).

The Convergence Theorem of the Perceptron algorithm states that in the case of noisy responses the condition $\mathfrak{p}(\mathbf{\Phi}_i)(\mathbf{u}.\mathbf{\Phi}_i) \geq 0$ cannot always be met. This condition relates the accuracy of the prediction performed by the Perceptron algorithm to the labels provided that afterwards update the respective hypothesis. In the case of noisy examples, we suggest that this condition should be modified so that it reflects the accuracy of the Perceptron algorithm in the presence of noise. Suppose that an example, such as $(\mathbf{\Phi}_i,\mathfrak{p}(\mathbf{\Phi}_i))$, is provided by EX_η. The probability that this example disagrees with any hypothesis $\mathbf{u}$ can be calculated as following:

$$\Pr[\mathfrak{p}(\mathbf{\Phi}_i)(\mathbf{u}\cdot\mathbf{\Phi}_i) < 0] \leq (1-\eta)\varepsilon + \eta(1-\varepsilon) < \eta_b + \varepsilon(1-2\eta_b) \tag{8}$$

From Eq. (8) it can be inferred that the expected rate of disagreement is at least η for the ideal hypothesis $\mathbf{u}$. Therefore, the separation factor of at least $\varepsilon(1-2\eta)$ should be between an ideal hypothesis and an approximation of that cf. [1]. As stated in the following theorem, the maximum number of mistakes that can be made by the Perceptron algorithm is polynomial in this separation.

Theorem 3. *Consider r labeled examples which are fed into the Perceptron algorithm, and let $\|\mathbf{\Phi}_i\| \leq R$. In the case of noisy labels, let $\mathbf{u}_n$ be the solution vector with $\|\mathbf{u}_n\| = 1$, and $\mathfrak{p}(\mathbf{\Phi}_i)(\mathbf{u}_n.\mathbf{\Phi}_i) \geq \varepsilon(1-2\eta_b) > 0$. Then*

$$N_{mis} = \left(\frac{R}{\varepsilon(1-2\eta_b)}\right)^2.$$

The key idea is that a separation factor of at least $\varepsilon(1-2\eta_b)$ must exist between $\mathbf{u}$ and $\mathbf{u}_n$. It is straightforward to prove this theorem, and for more details the reader is referred to [1].

Theorem 4. *When PAC learning the noisy XOR PUF, the maximum number of mistakes that the Perceptron algorithm can make is $N_{mis} = (n+1)^k/(\varepsilon^2(1-2\eta_b)^2)$. Furthermore, the maximum number of CRPs required for PAC learning a noisy XOR PUF is $O\big(\log(1/\delta)/(\varepsilon(1-2\eta_b)) + 4m^2(n+1)^k/(\varepsilon^3(1-2\eta_b)^3)\big)$.*

Proof. Following Corollary 2 and Theorem 3, this can be easily shown. ■

The most important message is that this maximum number of CRPs is polynomial in n, ε, δ as well as the upper bound of η. According to experimental results when the noise rate is 2 %, the number of CRPs required to learn a 128-bit XOR PUF ($k = 4$) is approximately increased by the factor 2, in comparison to the noiseless scenario with approximately the same ϵ [23]. For the same XOR PUF, increasing the noise rate to 5 % and 10 %, the number of CRPs is increased 2 times and 8 times, comparing with the case of $\eta_b = 0.02$. It has been concluded that the number of CRPs collected to model the XOR PUF is polynomial in the noise rate [23], which agrees with our theoretical result.

5 Discussion

5.1 Theoretical Considerations

By providing the proof of vulnerability of XOR PUFs to PAC learning we have demonstrated how fragile the security of this kind PUF is. The concept of PAC learning of XOR PUFs was almost catched by Hammouri et al. [10]. Although the authors benefit from one of the most adequate representations of the XOR PUFs, which is LTF-based [9], they could not prove the PAC learnability of the XOR PUFs. As the Vapnik-Chervonenkis dimension of an LTF representing an arbiter PUF is equal to $n+1$, this family of PUF primitives is subject to PAC learning attacks [7]. It is straightforward to further prove that the Vapnik-Chervonenkis dimension of the LTF representing an XOR PUF is $(n+1)^k$. Therefore, for constant k an XOR PUF with k chains (each with n stages) is also PAC learnable. In this paper, instead of sticking to this obvious fact, we introduced an algorithm that can PAC learn an XOR PUF, even in the case of noisy responses. However, the key argument supporting our claim is that the Vapnik-Chervonenkis dimension of the proposed LTF-based representation should be finite. Wrt. this argument, we have shown that by applying the Perceptron algorithm its run time is polynomial in n, $1/\varepsilon$, $1/\delta$, and k.

Another important aspect of our framework is the representation of an XOR PUF. As mentioned earlier, it is clear that according to what has been observed in [7], an XOR PUF can also be represented by a DFA with $O(n^k m^{2k})$ states. Therefore, their proposed algorithm makes $O\left(\left(1+2/\varepsilon \ln(1/\delta)\right)n^k M^{2k} + 2/\varepsilon n^{2k} M^{4k}\right)$ calls to EX. Comparing this number of calls with the number of calls that our algorithm makes to EX (see Corollary 2), it is clear that the numbers of calls made by both algorithms are polynomial in n, m, $1/\varepsilon$, and $1/\delta$. However, our algorithm outperforms in terms of the number of calls, and consequently its time complexity.

Of crucial importance for our framework is how the algorithm deals with noisy responses. In this paper we have proposed a model of noise, which is well-studied in the PAC learning related literature, and agrees with what can be seen in practice. Towards launching a machine learning attack, the adversary applies a set of challenges and collects the responses, where the latter might be noisy. From the lessons learnt from practice, the number of noisy response of an XOR PUF is virtually equal to the sum of the number of noisy response of each individual arbiter PUF [21]. In the literature majority voting is suggested as a solution to deal with noisy responses [18,23]. This can impair the performance of the proposed learning algorithm, when the attacker can observe each CRP only once and cannot do majority voting. It is even suggested that in order to reduce the effectiveness of ML attacks, the noise rate that can be observed by an attacker can be artificially increased, while the verifier still observes only a small noise rate [31]. In this latter scenario the majority voting cannot be helpful. On contrary, we have proved that XOR PUFs can be PAC learned without the need for majority voting.

We have stated that the maximum number of mistakes that the Perceptron algorithm can make, and consequently, the maximum number of CRPs required for PAC learning is polynomial in n, k, $1/\varepsilon$, $1/\delta$ as well as $1/(1-2\eta)$ in the case of noisy responses. Since we have mainly aimed to prove this, the maximum number of CRPs calculated in Sect. 3 ensures that the algorithm delivers an approximation of the desired hyperplane, with the probability at least $1-\delta$. The proposed upper bound of the number of CRPs can be improved to even reduce the number of required CRPs (see for instance [3]).

5.2 Practical Considerations

When proving that the Perceptron algorithm can be applied to PAC learn an XOR PUF, we take the advantage of the lessons learnt from practice, namely (a) the delay values can be mapped to a finite interval of integer values, and (b) the number of chains contained in an XOR PUF (k) cannot exceed a certain value. The importance of the first fact can be recognized in the recent proof of the PAC learnability of an XOR PUF (see Corollary 2). The second fact confirms that the Vapnik-Chervonenkis dimension of the LTF representing an XOR PUF is finite. Whereas the first fact has been already reflected in [7], the second one has been only partially discussed in the literature.

The results of experiments clearly demonstrate that XORing more than a certain number of chains is not feasible [21]. In their experiments, different XOR PUFs designed on 10 Xilinx Virtex 5 (LX110) FPGAs at the nominal condition (temperature = 35° C and VDD = 1V) are employed. For $k = 4$, it is reported that the noise rate is $\eta = 23.2\,\%$, and a change in the condition (e.g., reducing VDD) may result in an increase in the noise rate up to 43.2 %. Their most impressive achievement demonstrates that the noise rate of an XOR PUF is approximately equal to the sum of the noise rate of each individual arbiter PUF. For an arbiter PUF designed on 65 nm technology, a typical value of the noise rate is about 4 % [21]. Therefore, it can be approximated that the maximum of k can be ideally equal to 12, where the noise ratio would be approximately 50 %. Under this condition, even majority voting cannot be helpful so that the PUF cannot be verified. Another important factor limiting k is the silicon area used for constructing an XOR PUF. Based on a rough estimation reported in [16], the silicon area required for constructing an XOR PUF with k chains is k times larger than a single arbiter PUF.

Despite the implementation and technological limits on k, we have proved theoretical limits on when an XOR PUF can be learned in polynomial time. In practical studies it is not stated how the learnability is theoretically limited, even though the empirical upper bound reported in [16] and the experimental results in [23] are in line with our theoretical limit. Moreover, the experimental results presented in [30] are also evidences that support our findings. It has been shown that when $n = 64$ and $k \geq 4$, the number of CRPs required for the ML attack, and consequently the time complexity, is increased drastically. The same observation is repeated for $n = 128$ and $k \geq 5$. These emphasize the importance of our approach, in which not only the limit of the learnability in polynomial time is identified but also no side channel information is required to PAC learn the XOR PUFs under this limit. To evaluate the security of an XOR PUF with respect to this theoretical limit, the following scenarios can be distinguished:

- n is small (e.g., $n \leq 32$): in this case, the security can be easily broken by adopting a brute-force strategy.
- n is large (i.e., no brute-force strategy is applicable) and $k \gg (\ln n)$: under this condition, the XOR PUF cannot be learned in polynomial time. However, no practical implementation of such an XOR PUF is feasible due to the technological limits, more specifically the noisy responses.
- n is large and $k \ll (\ln n)$: the XOR PUF can be PAC learned.

In the latter scenario, it can be thought that an increase in m may lead to a more secure XOR PUF. Neither is this a valid theoretical approach nor it is possible in practice. From a theoretical point of view, although more CRPs are required for PAC learning an XOR PUF with large m, the number of CRPs is still polynomial in m, n and levels of accuracy and confidence. On the other hand, from a practical perspective, a chip designed with the large σ neither might work properly nor it can be utilized as a general purpose device [7]. Moreover, it can be suggested to produce arbiters with high precision in order to enlarge m. In this case, the cost of the chip is increased dramatically.

In previous studies, e.g., [23,24], powerful and costly machines have been employed to prove the concept of learnability of XOR PUFs. It might not be convenient to run a ML algorithm on such machines, particularly for XOR PUFs with large k and n. Since our concrete proofs state how the security of XOR PUFs can be broken in polynomial time, it seems redundant to conduct a simulation or an experiment concerning this issue. Last but not least, we emphasize that protocols relying on the security of XOR PUFs cannot be considered as an ultimate solution to the issue of insecure arbiter PUFs. As it has been also stated in [5], none of the XOR PUF-based protocols in its current form can be thought of as being perfectly secure.

6 Conclusion

We have developed a PAC learning framework, which clearly states how an XOR PUF can be learned, for given levels of accuracy and confidence. Furthermore, a theoretical limit for ML attacks as a function of the number of the chains and the number of arbiter PUF stages has been established. Moreover, we have proved that the maximum number of CRPs required for our framework is polynomial in the number of arbiter PUF stages, the pre-defined level of accuracy and confidence. It is further shown that our approach deals with the noisy responses in an efficient fashion so that in this case, the maximum number of CRPs collected by the attacker is polynomial in the noise rate. Our rigorous mathematical approach matches the results of experiments, which can be found in the literature. The observation made to reveal the technological limits on the number of chains contributes to the proof of vulnerability of XOR PUFs to PAC learning attacks.

Last but not least, on the basis of learning theory concepts, this study explicitly states that the current form of XOR PUFs cannot be considered as an ultimate solution to the problem of insecure arbiter PUFs. Furthermore, we believe that this work can provide an insight not only into the academic research but also for the design and manufacturing of delay-based PUFs.

References

1. Angluin, D., Laird, P.: Learning from noisy examples. Mach. Learn. **2**(4), 343–370 (1988)
2. Anthony, M.: Computational Learning Theory. Cambridge University Press, Cambridge (1997)
3. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K.: Learnability and the Vapnik-Chervonenkis dimension. J. ACM **36**(4), 929–965 (1989)
4. Bylander, T.: Learning linear threshold functions in the presence of classification noise. In: Proceedings of the Seventh Annual Conference on Computational Learning Theory, pp. 340–347 (1994)
5. Delvaux, J., Gu, D., Schellekens, D., Verbauwhede, I.: Secure lightweight entity authentication with strong PUFs: mission impossible? In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 451–475. Springer, Heidelberg (2014)
6. Freund, Y., Schapire, R.E.: Large margin classification using the perceptron algorithm. Mach. Learn. **37**(3), 277–296 (1999)
7. Ganji, F., Tajik, S., Seifert, J.P.: PAC Learning of Arbiter PUFs, Security Proofs for Embedded Systems-PROOFS (2014). https://eprint.iacr.org/2015/378.pdf. Accessed 18 May 2015
8. Gassend, B., Clarke, D., Van Dijk, M., Devadas, S.: Silicon physical random functions. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 148–160 (2002)
9. Gassend, B., Lim, D., Clarke, D., Van Dijk, M., Devadas, S.: Identification and authentication of integrated circuits. Concurrency Comput. Pract. Experience **16**(11), 1077–1098 (2004)
10. Hammouri, G., Öztürk, E., Sunar, B.: A tamper-proof and lightweight authentication scheme. Pervasive Mobile Comput. **4**(6), 807–818 (2008)
11. Khardon, R., Wachman, G.: Noise tolerant variants of the perceptron algorithm. Journal Mach. Learn. Res. **8**, 227–248 (2007)
12. Kömmerling, O., Kuhn, M.: Design principles for tamper-resistant security processors. In: USENIX Workshop on Smartcard Technology (1999)
13. Lee, J.W., Lim, D., Gassend, B., Suh, G.E., Van Dijk, M., Devadas, S.: A technique to build a secret key in integrated circuits for identification and authentication applications. In: Symposium on VLSI Circuits, 2004. Digest of Technical Papers, pp. 176–179 (2004)
14. Littlestone, N.: Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach. Learn. **2**(4), 285–318 (1988)
15. Littlestone, N.: From on-line to batch learning. In: Proceedings of the Second Annual Workshop on Computational Learning Theory, pp. 269–284 (1989)
16. Maes, R.: Physically Unclonable Functions: Constructions, Properties and Applications. Springer, Heidelberg (2013)

17. Maes, R., Verbauwhede, I.: Physically unclonable functions a study on the state of the art and future research directions. In: Sadeghi, A.-R., Naccache, D. (eds.) Towards Hardware-Intrinsic Security. Information Security and Cryptography, pp. 3–37. Springer, Heidelberg (2010)
18. Majzoobi, M., Koushanfar, F., Devadas, S.: FPGA PUF using programmable delay lines. In: 2010 IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6 (2010)
19. Majzoobi, M., Koushanfar, F., Potkonjak, M.: Lightweight secure PUFs. In: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, pp. 670–673 (2008)
20. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science **297**(5589), 2026–2030 (2002)
21. Rostami, M., Majzoobi, M., Koushanfar, F., Wallach, D., Devadas, S.: Robust and reverse-engineering resilient puf authentication and key-exchange by substring matching. IEEE Trans. Emerg. Top. Comput. **2**(1), 37–49 (2014)
22. Ruhrmair, U., Solter, J., Sehnke, F., Xu, X., Mahmoud, A., Stoyanova, V., Dror, G., Schmidhuber, J., Burleson, W., Devadas, S.: PUF modeling attacks on simulated and silicon data. IEEE Trans. Inf. Forensics Secur. **8**(11), 1876–1891 (2013)
23. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling attacks on physical unclonable functions. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 237–249 (2010)
24. Rührmair, U., Xu, X., Sölter, J., Mahmoud, A., Majzoobi, M., Koushanfar, F., Burleson, W.: Efficient power and timing side channels for physical unclonable functions. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 476–492. Springer, Heidelberg (2014)
25. Sadeghi, A.R., Naccache, D. (eds.): Towards Hardware-Intrinsic Security: Foundations and Practice, 1st edn. Springer, Heidelberg (2010)
26. Servedio, R.A.: Efficient Algorithms in Computational Learning Theory. Harvard University, Cambridge (2001)
27. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, Cambridge (2014)
28. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th Annual Design Automation Conference, pp. 9–14 (2007)
29. Tajik, S., Dietz, E., Frohmann, S., Seifert, J.-P., Nedospasov, D., Helfmeier, C., Boit, C., Dittrich, H.: Physical characterization of arbiter PUFs. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 493–509. Springer, Heidelberg (2014)
30. Tobisch, J., Becker, G.T.: On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation (2015). https://www.emsec.rub.de/research/publications/ScalingPUFCameraReady/. Accessed 18 May 2015
31. Yu, M.D.M., Verbauwhede, I., Devadas, S., MRaihi, D.: A noise bifurcation architecture for linear additive physical functions. In: 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 124–129 (2014)

Springer

http://www.springer.com/978-3-319-22845-7

Trust and Trustworthy Computing
8th International Conference, TRUST 2015, Heraklion,
Greece, August 24-26, 2015, Proceedings
Conti, M.; Schunter, M.; Askoxylakis, I. (Eds.)
2015, XI, 328 p. 79 illus., Softcover
ISBN: 978-3-319-22845-7