

PAC Learning of Arbiter PUFs

Fatemeh Ganji, Shahin Tajik, Jean-Pierre Seifert

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

Abstract. The general concept of Physically Unclonable Functions (PUFs) has been nowadays widely accepted and adopted to meet the requirements of secure identification and key generation/storage for cryptographic ciphers. However, shattered by different kinds of attacks, it has been proved that the promised security features of arbiter PUFs, including unclonability and unpredictability do not hold unconditionally. It has been stated in the literature that arbiter PUFs can be broken by launching modeling attacks, i.e., applying machine learning methods. In this case, a large set of Challenge-Response Pairs (CRP) is collected by an attacker, aiming at mathematically modeling the Challenge-Response behavior for a given arbiter PUF. However, the success of all existing modeling attacks rests so far on pure *trial* and *error* estimates. This means that neither the probability of obtaining a useful model (confidence), nor the maximum number of required CRPs, nor the correct prediction of an unknown challenge (accuracy) is guaranteed at all. To address these issues, this work will present a Probably Approximately Correct (PAC) learning algorithm. This proves that learning of any arbiter PUF for prescribed levels of accuracy and confidence can be done in polynomial time. Based on some crucial discretization process, we are able to define a Deterministic Finite Automaton (of polynomial size), which exactly accepts that regular language that corresponds to the challenges mapped by the given PUF to *one* responses. We also prove that the maximum required number of CRPs is polynomial in the number of arbiter stages. A further analysis reveals that this maximum number of CRPs is also polynomial in the maximum deviation of the arbiter delays as well as the pre-defined levels of accuracy and confidence. To the best of our knowledge this is the first algorithm which can provably learn an arbitrary arbiter PUF. Moreover, our proof of the PAC learnability of arbiter PUFs gives many new insights into the correct construction of secure arbiter PUFs in general.

Keywords: Arbiter PUF, Deterministic Finite Automata, Machine Learning, PAC Learning, Regular Language

1 Introduction

Physically Unclonable Functions (PUFs), as introduced by Gassend et al. [6], have attracted a great deal of research interest due to their potential enhancement of numerous applications. These applications are ranging from hardware

fingerprinting and authentication [27, 31] to secure storage for encryption mechanisms [13, 28]. Arbiter PUFs are one type of PUF instantiations [15], where the core idea is to exploit the delay differences between symmetrically designed electrical paths on a silicon chip to generate a somehow random but unique response [12].

Similar to all kind of PUFs, *unclonability* and *unpredictability* are the main requirements of the arbiter PUF family [3, 23]. However, contrary to these basic requirements, previous work in the literature introduced different successful attacks on arbiter PUFs. The attacks on arbiter PUFs can be classified into two categories: side channel attacks and modeling (i.e., machine learning) attacks. The former type of attack uses the side channel information, such as photonic emissions and electromagnetic radiations, to physically characterize an arbiter PUF [30, 5, 4]. On the other hand, modeling attacks require only a subset of Challenge-Response Pairs (CRPs) to build a mathematical model of the arbiter PUF, which later can predict the response of that arbiter PUF with some probability [12, 25]. As being non-invasive, modeling attacks can be more cost and time effective in comparison to side channel attacks.

Although an increase in the number of PUF stages or the addition of non-linearity to the PUF response can reduce the effectiveness of modeling attacks [13, 29], by using extra side channel information and utilizing more advanced machine-learning tools (i.e., requiring a larger subset of CRPs) an attacker can still break the security of such arbiter PUFs [26, 16]. It has been verified experimentally that the number of CRPs required for a successful attack increases exponentially with the number of stages [16]. This raises the natural question to what preciseness an attacker in general can model an arbiter PUF with an arbitrary number of stages. In other words, how many CRPs are really required to model the PUF for given levels of *accuracy* and final model delivery *confidence*. Unfortunately, this issue has not been solved at all in the literature so far and thus modeling attacks rely only on trial and error or heuristic approaches.

In this paper we present a novel well-defined mathematical representation of arbiter PUFs. Based on this representation, we introduce a polynomial-time learning algorithm that provably learns the challenge-response behavior of an arbitrary arbiter PUF for given levels of accuracy and confidence. We will show how the levels of accuracy and confidence of a model are related to the number of collected CRPs and the number of stages of an arbiter PUF, as well as the maximum variation of delay values. In contrast to what has been claimed in the literature so far, we will prove that the maximum number of CRPs required for the attack is polynomial in the number of stages. Finally, we evaluate the time complexity of our learning algorithm and prove that it is polynomial in the length of a given arbiter PUF, its maximum variation of delays and levels of accuracy and confidence. The main contributions of this paper are as follows:

A learning algorithm for prescribed levels of accuracy and confidence. Based on a new mathematical representation for arbiter PUFs, we introduce a learning algorithm, which is able to model the PUF for given levels of accuracy and confidence. The proposed algorithm can be applied to break

the security of different types of arbiter PUFs, including non-linear PUFs (e.g., XOR arbiter PUFs).

Discretization of real-valued delay values. Our mathematical representation is based on defining a set of proper *integer* delay values that, at first glance, are contrary to the *real* delay values, which can be observed on a chip. However, we will first explain that the statistical relevant delay values of an arbiter PUF are distributed in a limited interval. Secondly, it will be demonstrated how we can discretize the different delay values with regard to the limited precision of the arbiter placed at the end of its respective chain. Finally, due to these facts, we present a mapping between these discrete delay values to a fitting set of integer values.

Calculation of the maximum number of CRPs required for our new attack. In order to learn the challenge-response behavior of an arbiter PUF for given levels of accuracy and confidence, we prove that the maximum number of CRPs required for launching our attack is polynomial in the length of an arbiter PUF. Besides that, the impact of the limited variation of the delays on the learnability of an arbiter PUF is also discussed.

Evaluation of the time complexity of the new attack. Finally, we will evaluate the time complexity of our learning algorithm. Our proofs reveal that the running time of the proposed learning algorithm is polynomial in the length of the given arbiter PUF, the maximum variation of delays and the given levels of accuracy and confidence.

2 Notation and preliminaries

This section provides basic background and notations required to understand the present paper. Particular emphasis is given to arbiter PUFs, regular languages, DFAs, and PAC learning. Unfortunately, all these different fields have their own established nomenclature which we kept in each of the following sections for the sake of simplicity. However, since our framework covers all these individual fields, caution and flexibility regarding the notations used in more than one fields are required by the reader.

2.1 Arbiter PUFs

PUFs are physical input to output mappings, which are most often entangled with the intrinsic silicon properties of a chip. The input and output of a PUF are called *challenge* and *response*, respectively. A PUF can be described by the function $f_{\text{PUF}} : \mathcal{C} \rightarrow \mathcal{Y}$ where $f_{\text{PUF}}(c) = y$, $\mathcal{C} = \{0,1\}^n$ being the set of challenges, and $\mathcal{Y} = \{0,1\}$ the set of responses, c.f. [15]. PUFs in general have a set of crucial properties, including being evaluable, unique, unclonable, and one-way, c.f. [15].

Arbiter PUFs are one of the major classes of the PUFs. An arbiter PUF consists of multiple switch blocks, so called *stages*, connected in a chain that ends in an arbiter, see Figure 2. In our notation, n is the number of stages of an

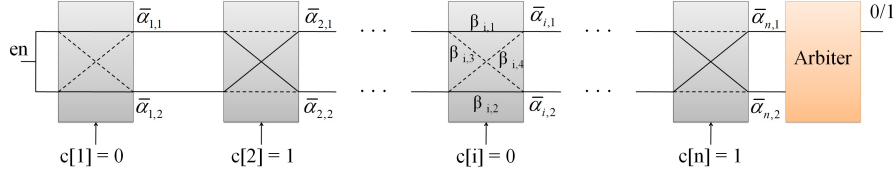


Fig. 1: Schematic of an arbiter-PUF

arbiter PUF under consideration. A challenge is a string $c = c[1] \cdots c[n]$ of n bits, where each bit (e.g., $c[i]$) is fed into a single stage (e.g., i^{th} stage). The signal propagates through the direct paths inside the i^{th} stage if $c[i] = 0$, otherwise the crossed paths are utilized. Let \mathcal{B}_i denote a random variable related to the delay within the i^{th} stage. The realizations of the variables \mathcal{B}_i in an arbiter PUF are certain $\beta_{i,1}$, $\beta_{i,2}$, $\beta_{i,3}$, and $\beta_{i,4}$. Here $\beta_{i,1}$ and $\beta_{i,2}$ are the delays of the upper and lower direct paths in the i^{th} stage, respectively, see Figure 2. On the other hand, the delays of the upper and lower crossed paths in the i^{th} stage are $\beta_{i,3}$ and $\beta_{i,4}$, respectively. \mathcal{B}_i follows a Gaussian distribution with the mean μ_i and the deviation ω_i , cf. [24].

We define \mathcal{A}_i as a random variable which corresponds to the total delay between the enable point and the outputs of the i^{th} stage of the PUF. Following the linear additive model of the arbiter PUF, cf. [12], we have

$$\mathcal{A}_i = \sum_{k=1}^i \mathcal{B}_k.$$

The realizations of the partial sums \mathcal{A}_i at the outputs of the i^{th} stage are denoted by $\bar{\alpha}_{i,j}$, where j represents the upper and lower output (i.e., $j = 1$ for upper and $j = 2$ for lower output), see Figure 2. The arbiter at the end of the PUF chain has a precision $\gamma > 0$, and compares the arrival times of signals on the upper and lower paths (i.e., $\bar{\alpha}_{n,1}$ and $\bar{\alpha}_{n,2}$). More formally, we assume that the output of the arbiter is “1” if $\bar{\alpha}_{n,1} - \bar{\alpha}_{n,2} > \gamma$, whereas it is “0” if $\bar{\alpha}_{n,2} - \bar{\alpha}_{n,1} > \gamma$. The metastable condition, where $|\bar{\alpha}_{n,1} - \bar{\alpha}_{n,2}| < \gamma$, will be discussed later in Section 5. Finally, in order to take into account the impact of different path configurations for $\bar{\alpha}_{n,1}$ and $\bar{\alpha}_{n,2}$, we define the single bit

$$u_i = \bigoplus_{k=1}^i c[k]$$

related to the history of the paths that the signal follows.

2.2 Regular Language and Principles of DFAs

We assume that the reader is familiar with regular languages and Deterministic Finite Automata (DFA). Therefore, we will only briefly recall the notations, which will be used throughout this paper. We follow the standard notation, as found in [1] and [9]. Consider the alphabet $\Sigma = \{0,1\}$ and the set of all strings

Σ^* over Σ . By $|c|$ we denote the length of strings $c \in \Sigma^*$, and by λ the empty string of length $|\lambda| = 0$.

A DFA A is given by $A = (Q, \delta, \Sigma, q_0, F)$ over the alphabet Σ , with Q being the set of states, the initial state q_0 , and the accepting states $F \subseteq Q$. The transition function $\delta : Q \times \Sigma \rightarrow Q$ is defined as follows. For all $q \in Q$, $a \in \Sigma$ and $c \in \Sigma^*$ we have $\delta(q, \lambda) = q$ and its canonical continuation to Σ^* , i.e., $\delta(q, ac) = \delta(\delta(q, a), c)$. The set of strings accepted by A is called its accepted language $L(A) := \{c \in \Sigma^* \mid \delta(q_0, c) \in F\}$, i.e., a regular language. A state q_i is *live*, if there exist $c_1, c_2 \in \Sigma^*$ such that $c_1 c_2 \in L(A)$ with $\delta(q_0, c_1) = q_i$ and $\delta(q_i, c_2) \in F$. Otherwise, q_i is called dead.

2.3 The PAC Model

The idea of the PAC (Probably Approximately Correct) model is that successful learning of an unknown target should yield, with high probability, a hypothesis that is a good approximation of the unknown target. In order to describe this idea more precisely, we recall now the basic definitions of the PAC model. For a more thorough introduction into this model and the PAC learning framework, the reader is referred to the excellent textbook [11].

Let \mathfrak{C}_n be a *target concept class* over the *instance space* $X_n = \{0, 1\}^n$, and let $X = \cup_{n \geq 1} X_n$ and $\mathfrak{C} = \cup_{n \geq 1} \mathfrak{C}_n$. \mathfrak{H}_n as the *hypothesis space* and \mathfrak{H} are defined similarly. The notion of approximation is defined by assuming that there is an arbitrary probability distribution D on the instance space X_n , giving the probability of each instance. We then define the *error* of a hypothesis $h \in \mathfrak{H}$ with respect to a fixed target concept $c \in \mathfrak{C}$, denoted by $error(h)$, as

$$error(h) := \sum_{x \in h \Delta c} D(x),$$

where Δ is the symmetric difference. The concept class \mathfrak{C} is called PAC learnable by the hypothesis space \mathfrak{H} if there exists a polynomial time algorithm \mathcal{L} and a polynomial $p(\cdot, \cdot, \cdot)$ such that for all $n \geq 1$, all target concepts $c \in \mathfrak{C}_n$, all distributions D on the instance space X_n , and all ε and δ , with $0 < \varepsilon, \delta < 1$, the following holds. If the algorithm L is given at least $p(n, 1/\varepsilon, 1/\delta)$ independent random samples of c drawn according to D , then with probability at least $1 - \delta$ the algorithm L returns a hypothesis $h \in \mathfrak{H}_n$ such that $error(h) \leq \varepsilon$. The smallest polynomial p is called the *sample complexity* of L . With respect to \mathfrak{C} and \mathfrak{H} , two cases are interesting. The first is, where $\mathfrak{C} = \mathfrak{H}$. In this case we say that \mathfrak{C} is *properly* PAC learnable. On the other hand, although the other case seems to be obvious, it should be more carefully considered. Indeed, if \mathfrak{C} is a concept class and there exists a hypothesis class \mathfrak{H} such that: *a)* \mathfrak{H} can be evaluated on given instances in polynomial time, and *b)* \mathfrak{C} is PAC learnable by \mathfrak{H} , then we eventually say that \mathfrak{C} is *PAC learnable*.

2.4 PAC Learning of a Regular Language

It has been proven by Angluin [1] that regular languages are PAC learnable by DFAs. A DFA (as defined above) is one possible representation of a regular

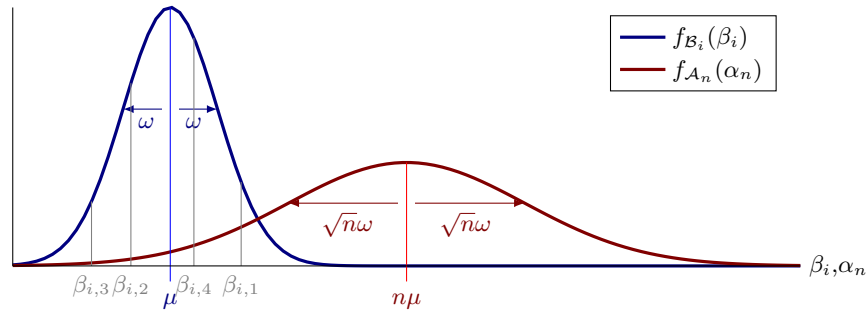


Fig. 2: The distribution of \mathcal{B}_i (blue) with the mean μ and deviation ω . Four examples of possible realization of \mathcal{B}_i are $\beta_{i,1}$, $\beta_{i,2}$, $\beta_{i,3}$, and $\beta_{i,4}$, which correspond to four delays at the i^{th} stage. The distribution of total propagation delay from the enable point to the outputs of last stage in an arbiter PUF with length n (red), with mean $n\mu$ and deviation $\sqrt{n}\omega$.

language (i.e., the target concept) that can be learned by Angluin’s algorithm. The learning algorithm proposed in [1] efficiently learns an unknown regular set from an adequate teacher. For an arbitrary conjecture, the original algorithm calls the teacher to query whether the conjecture is correct. In the case that it is not correct, the teacher provides a counterexample.

Of crucial importance for our purposes is the fact that Angluin also proved that her original algorithm can be modified to obtain an algorithm for PAC learning regular languages. In this case, instead of providing counterexamples to the learning algorithm, it gets access to an Oracle EX , providing labelled examples. For a given input c the oracle $EX : \Sigma^* \rightarrow \Sigma$ returns a label $\mathbf{p}(c)$ identifying whether the string c belongs to the regular language L under consideration. Hence for all $c \in L$, we have $\mathbf{p}(c) = 1$, and otherwise $\mathbf{p}(c) = 0$. A labeled example is the pair $(c, \mathbf{p}(c))$ and the set of positive examples (i.e., $\mathbf{p}(c) = 1$) is denoted by S^1 , whereas the set of negative examples is S^0 .

3 Representing Arbiter PUFs by DFAs

In this section we will derive a polynomial-sized representation of arbiter PUFs by DFAs. To this end, we first show how the real delay values of an arbiter PUF can be *mapped* to a finite set of integer values. On the basis of this mapping, we introduce a DFA-based representation of an arbiter PUF.

3.1 Discretization Process of Delay Values

As mentioned before, the delay differences in the stages of an arbiter PUF are caused by variations in the manufacturing processes. It has been shown that \mathcal{B}_i (the aforementioned random variable describing the delay of the i^{th} stage)

follows a Gaussian distribution, c.f. [24]. Thus, $\mathcal{B}_i \sim N(\mu_i, \omega_i)$ and its Probability Density Function (PDF) $f_{\mathcal{B}_i}(\beta_i)$ is given by

$$f_{\mathcal{B}_i}(\beta_i) = \frac{1}{\omega_i \sqrt{2\pi}} e^{-(\beta_i - \mu_i)^2 / 2\omega_i^2}.$$

The mean μ_i is often reported by manufactures as the nominal propagation delay of the utilized multiplexer, and the standard deviation ω_i is caused by the variations in the manufacturing process, c.f. [20]. As 99.7% of a gaussian distribution lie within the range of three standard deviation away from the mean value, the realizations of the \mathcal{B}_i for the i^{th} stage (i.e., $\beta_{i,1}$, $\beta_{i,2}$, $\beta_{i,3}$, and $\beta_{i,4}$) are drawn from an interval, whose length is 6ω , see Figure 2.

By the additive linear model of the arbiter PUF we have $\mathcal{A}_i = \sum_{k=1}^i \mathcal{B}_k$, where \mathcal{A}_i is the random variable, which shows the total propagation delays at the outputs of the i^{th} stage. Therefore, the PDF of the total propagation delays at the outputs of each stage are the convolution of all PDFs of the previous stages [21], i.e., $f_{\mathcal{A}_i}(\bar{\alpha}_i) = f_{\mathcal{B}_i}(\beta_i) * f_{\mathcal{B}_{i-1}}(\beta_{i-1}) * \dots * f_{\mathcal{B}_1}(\beta_2) * f_{\mathcal{B}_1}(\beta_1)$. As all delays on each stage follow the normal distribution, \mathcal{A}_i also follows the normal distribution. Hence, in an arbiter PUF of length n , if we assume that $\mu_1 = \mu_2 = \dots = \mu_n = \mu$ and $\omega_1 = \omega_2 = \dots = \omega_n = \omega$, we can write

$$f_{\mathcal{A}_i}(\bar{\alpha}_i) = \frac{1}{i\omega \sqrt{2\pi}} e^{-(\bar{\alpha}_i - i\mu)^2 / 2i\omega^2}.$$

As a result, the random variable \mathcal{A}_n , corresponding to the total propagation delays until the last stage, will have the mean $n\mu$ and the standard deviation $\sqrt{n}\omega$, see Figure 2. Therefore, we can assume that all statistically relevant delay values will lie in a limited interval, whose length is $6\sqrt{n}\omega$ (i.e. within three standard deviation away from the mean value $n\mu$).

It is obvious that the delay differences are real numbers. However, the arbiter at the end of the chain provides only a limited precision in terms of comparing the total propagation delays of two paths [18]. Hence, it can compare two signals with delay differences only above a certain threshold, say $\gamma > 0$. As a result, the actual number of different delay values, which can be observed and compared by the arbiter are limited to

$$M = \left\lceil \frac{6\sqrt{n}\omega}{\gamma} \right\rceil.$$

Due to this fact, all propagation delays at the output of each stage (i.e., $\bar{\alpha}_{i,j}$, where $0 \leq i \leq n$ and $1 \leq j \leq 2$) can be mapped to integer values between 0 and M , corresponding to the minimum and maximum real values, respectively. The mapped values are denoted by $\alpha_{i,j}$, and therefore, we have $\alpha_{i,j} \in \mathbb{Z}$. In this case, the response of the arbiter is “1” if $\alpha_{n,1} - \alpha_{n,2} > 1$, whereas it is “0” if $\alpha_{n,2} - \alpha_{n,1} > 1$. The arbiter is in the metastable condition, if $|\alpha_{n,1} - \alpha_{n,2}| = 0$.

3.2 Building a DFA out of an Arbiter PUF

According to the definition of a PUF, as presented in Section 2.1, it seems tempting to represent PUFs by boolean functions. However, being physical instances,

boolean functions cannot concisely capture the physical characteristics of a PUF. Still, we can evaluate a PUF in polynomial time (see Section 2.1). However, in order to PAC-learn the intrinsic challenge-response behavior of a given PUF, a polynomial-size representation of its behavior is required. Therefore, we aim to derive such a concise representation that can be used in a second step to provide a PAC-learning algorithm, which works in polynomial time and learns the unknown challenge-response behavior of an arbiter PUF for predefined levels of accuracy and confidence.

Consider a PUF, whose challenge-response functionality is given by the mapping $f_{\text{PUF}} : \mathcal{C} \rightarrow \mathcal{Y}$ where $f_{\text{PUF}}(c) = y$, \mathcal{C} is the set of challenges, and \mathcal{Y} is the set of responses, c.f. [15]. For an arbiter PUF we of course have $\mathcal{C} = \{0,1\}^n$ and $\mathcal{Y} = \{0,1\}$. Let us define $L_{f_{\text{PUF}}} := \{c \in \mathcal{C} \mid f_{\text{PUF}}(c) = 1\}$. We have $L_{f_{\text{PUF}}} \subseteq \{0,1\}^n \subseteq \Sigma^*$, where $\Sigma = \{0,1\}$. Hence, $L_{f_{\text{PUF}}}$ can be thought as being the accepted language of a certain automaton. It accepts those strings $c \in \mathcal{C}$, whose length is n and $f_{\text{PUF}}(c) = 1$.

In order to build an automaton A , we will use the notation of the “integer” PUF propagation delays $\alpha_{i,j}$, see Section 3.1. See Figure 3 for the central idea of our automaton construction. After reading the first challenge bit applied to the first stage, A transits from q_0 (the initial state) to either $q_{1,1}$ or $q_{1,2}$, depending on whether $c[1] = 0$ or $c[1] = 1$. Here $q_{1,1}$ and $q_{1,2}$ correspond to tuples $((\alpha_{1,1}, \alpha_{1,2}), u_1, i = 1)$, where the first elements of them are ordered pairs of possible delays at the output of the first stage. Note that due to the physical characteristics of the arbiter PUF — after the signal propagated through either the direct or crossed path — two different pairs of delays will be obtained. These two pairs are included in $q_{1,1}$ and $q_{1,2}$. The second element of these tuples is u_1 which “memorizes” the first i^{th} bits of the challenge, for the first stage we have $i = 1$ and thus, $u_1 = c[1]$. Finally, the third element of these tuples represents the depth in which the respective nodes are, e.g., $i = 1$ for $q_{1,1}$ and $q_{1,2}$. This component “counts” the number of bits of the input c , which have been consumed so far.

In order to further elaborate on the definition of $q_{i,j}$, we provide an example on how the DFA transits from $q_{i,j}$ to $q_{i+1,j'}$ when reading $c[i+1] = 0$, c.f. Figure 3. As it can be seen from the figure, when $u_{i+1} = 0$, the first entry of the pair $(\alpha_{i+1,1}, \alpha_{i+1,2})$ contains the delays on the upper paths (i.e., $\alpha_{i+1,1} = \alpha_{i,1} + \beta_{i+1,1}$), whereas for $u_{i+1} = 1$ it is composed of the delays on the lower paths, i.e., $\alpha_{i+1,1} = \alpha_{i,2} + \beta_{i+1,2}$. Hence, using u_{i+1} , the correct sum is defined. What has been explained here is applicable for the other stages of the arbiter PUF as well, the other states of A can be defined similarly.

The further and crucial characteristics of this DFA representing an arbiter PUF are as follows. In Figure 3, the states $q_{n,1}, \dots, q_{n,2^n}$ define the *possible* accepting states in the following way. As mentioned in Section 3.1, assume that $f_{\text{PUF}}(c) = 1$ if $\alpha_{n,1} - \alpha_{n,2} > 1$, and when $\alpha_{n,2} - \alpha_{n,1} > 1$, $f_{\text{PUF}}(c) = 0$. For a challenge string $c = c_1 c_2 \dots c_n$, A accepts c if there is a sequence of states such that $r_0 = q_0$, $r_i = \delta(r_{i-1}, c_i)$ with $1 \leq i \leq n$ and $r_n \in F$, where

$$F = \{q_{n,k} \mid q_{n,k} = ((\alpha_{n,1}, \alpha_{n,2}), u_n, n) \text{ s.t. } \alpha_{n,1} - \alpha_{n,2} > 1, 1 \leq k \leq 2^n\}.$$

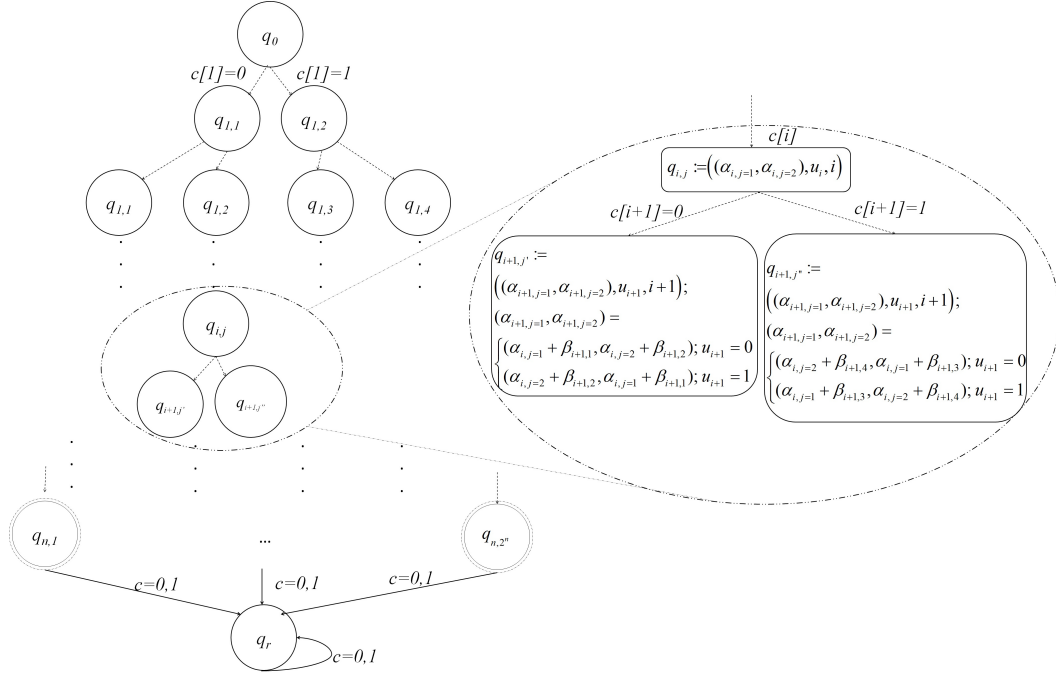


Fig. 3: A DFA representing an arbiter PUF.

All other states, not being defined as accepting states, are of course rejecting. Although it is possible that $\alpha_{n,1} = \alpha_{n,2}$, in reality it will almost never occur (see Section 5 for more details), thus we can safely exclude this case. The special rejection state q_r is reached after reading further bits following the n^{th} bit of a given input. After reaching the rejecting state, reading any further bit results in staying in q_r . We should stress that since challenges are n -bit strings, all longer strings are rejected, as well as shorter strings.

As it is evident from the above discussion, the size of A as constructed above and based upon the total delay at the output of each stage of the arbiter PUF is clearly exponential in n . Consequently, if we represent an arbiter PUF by this A , the output of a learning algorithm should be a hypothesis h with $h \in \mathfrak{H}_n$ that could not be evaluated in polynomial time at all. However, having a closer look, it will be proved that this DFA has indeed only a size which is polynomial in n , and can be used to PAC-learn an arbiter PUF. To shrink A , we use the results of the discretization process of the real delay values.

As mentioned in Section 3.1, the total delay values can be mapped to the finite set of integer values in $[0, M]$, where M can be regarded as a constant independent of n . Therefore, the number of possible values of $\alpha_{i,j}$ for $1 \leq i \leq n$ and $1 \leq j \leq 2$ is $M + 1$. For a given depth i , the number of possible pairs of delays on the upper and lower paths (i.e., $(\alpha_{i,1}, \alpha_{i,2})$) is thus at most $(M + 1)^2$. Consequently, the number of distinguishable pairs, i.e., corresponding to the

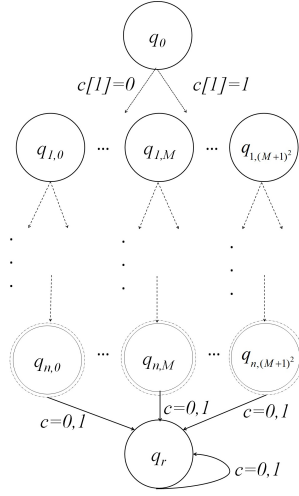


Fig. 4: The shrunk DFA representing an arbiter PUF. Note that the size of this DFA is clearly polynomial in n .

different ordered pairs of sums in each level of A cannot exceed $(M + 1)^2$, and thus the total number of distinguishable states is limited by

$$O(n(M + 1)^2).$$

Collapsing the indistinguishable states, a much smaller DFA as shown in Figure 4 is obtained.

4 PAC-learning of Arbiter PUFs

With the help of the polynomial-size DFA derived in Section 3, we can now describe a PAC-learning algorithm to efficiently learn the challenge-response behavior of a given arbiter PUF. Such a PAC-learning algorithm can be derived by adopting and modifying the algorithms presented in the literature [1, 11, 22]. The algorithm presented later by us can be seen as an adapted version of what has been proposed by Angluin [1]. Here we only describe the algorithm briefly, and refer the reader to [1] for further details.

A given PUF provides the learner with access to the Oracle $EX := f_{\text{PUF}}$. See Section 2.4 and Figure 5. The oracle EX provides labelled examples, whose length are exactly n . Moreover, the length n of the examples, the maximum delay value M , and the levels for the accuracy and confidence are provided also as inputs to the algorithm.

The main steps of the PAC-learning algorithm are depicted in Table 1. At the first stage, h_0 contains λ and no string c is accepted. Since all examples with $f_{\text{PUF}}(c) = 0$ are rejected, and h_0 will be modified only after receiving a positive example, we assume wlg. that the first example is positive. For the

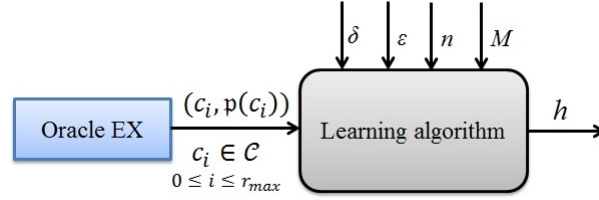


Fig. 5: Block diagram of the PAC-learning algorithm

i^{th} example, the algorithm examines whether h_{i-1} is consistent. Then h_{i-1} is updated, if it is not consistent with the example. The procedures of checking the consistency and updating a hypothesis are described extensively in [1] and [11], and are not further discussed by us. Moreover, the proof of the correctness of the above algorithm \mathcal{L} is also presented in [1], and the reader is again referred to her celebrated result. The algorithm ensures that \mathcal{L} makes at most r_{max} calls to the oracle EX , and that the final output h is an $\varepsilon/2$ -approximation of S^1 , as shown by the following theorem.

Theorem 1. *Let $N := O(nM^2)$ that represents the number of live states, then \mathcal{L} returns a hypothesis h after at most $O(N + (1/\varepsilon)(N \log(1/\delta) + N^2))$ calls to EX , and with probability at least $(1 - \delta/2)$, h is an $\varepsilon/2$ -approximation of S^1 .*

As all the details of the proof are already given by Angluin [1], we elaborate only on a few interesting points from the proof given by Angluin. It is clear that her algorithm is also applicable in the case that the oracle EX , as in our case, provides only examples of length n . This means that the PAC-learning problem addressed by us can be thought of as being a simplified version of the more general problem solved in [1]. Furthermore, our algorithm is designed to learn exactly those challenges yielding a 1 at the output of the final arbiter of the given PUF.

The proof is based on the maximum number of calls that \mathcal{L} makes to EX . We can calculate the maximum number of calls r_{max} as follows. As at most $(N - 1)$ different h_k have to be tested, we have

$$\begin{aligned}
 r_{max} &= \sum_{i=0}^{(N-2)} (r_i + 1) \\
 &= (N - 1) + 2/\varepsilon \left((N - 1) \ln(1/\delta) + \ln(2) \sum_{i=0}^{(N-2)} (i + 1) \right)
 \end{aligned}$$

which results in

$$O \left(\left(1 + \frac{2}{\varepsilon} \ln(1/\delta) \right) n(M + 1)^2 + \frac{2}{\varepsilon} n^2 (M + 1)^4 \right).$$

Now we calculate the probability that the output h is not a $\varepsilon/2$ -approximation of S^1 . As in [1] we have

$$\Pr[\text{error}(h) > \varepsilon/2] = \sum_{i=0}^{(N-2)} (1 - \varepsilon/2)^{r_i}$$

which yields that

$$\Pr[\text{error}(h) \leq \varepsilon/2] \geq 1 - \frac{\delta}{2}.$$

The time consumed by \mathcal{L} is bounded by the time spent on searching for such strings, which distinguish two states in h_k . Since the maximum size of a set containing all possible strings is $|\Sigma| \cdot |A_{\text{PUF}}| + 1$, which is $2(n \cdot (M+1)^2) + 1$, and the related steps (step 4 to 8 in Figure 6) can be repeated at most r_{max} times, the time complexity of \mathcal{L} is bounded by

$$O\left(\left(1 + (2/\varepsilon) \ln(1/\delta)\right)n^2(M+1)^4 + (2/\varepsilon)n^3(M+1)^6\right).$$

Thus, we concluded that S^1 can be efficiently learned (for given accuracy and confidence levels), and the respective algorithm runs indeed in time polynomial in n , M , $1/\varepsilon$ and $1/\delta$.

Algorithm
Input: A list of $(c_i, f_{\text{PUF}}(c_i))$, (ε, δ) , n , and M
Output: h
begin
1: $h_0 := \lambda$
2: Wlog. let the first positive example be $(c_1, 1)$
3: $k = 0, i = 1$
4: Examine the consistency of h_k
5: if h_k is not consistent with c_i then
6: Update h_k
7: $k = k + 1$
8: fi
9: $i = i + 1$
10: Let $r_i := \lceil 2/\varepsilon(\ln(1/\delta) + (i+1) \ln 2) \rceil$
11: while $r_i \leq r_{max}$ do
12: Proceed with the next example
13: Do steps 4-8
14: od
15: $h := h_k$
end

Table 1: PAC-learning algorithm \mathcal{L} for f_{PUF} .

5 Practical Considerations

We have proved that the arbiter PUF family is subject to PAC-learning attacks and presented an algorithm, which predicts the output of an arbiter PUF for given ε and δ . The notion of PAC-learning was already earlier used in the context of PUFs by Hammouri et al. [8] to assess the security of their proposed scheme. In order to do so, they represent a noiseless arbiter PUF as a Linear Threshold

Function (LTF) [2]. The LTF-model for PUFs is actually due to [7]. This implicitly leads to the conclusion that a noiseless arbiter PUF is PAC-learnable, as the Vapnik–Chervonenkis dimension of the proposed representation (LTF) for a noiseless arbiter PUF is equal to $n + 1$. Consequently, a noiseless arbiter PUF is in principle PAC-learnable under LTF-based representations. For more details and a proof of this, c.f. [2]. Nevertheless, to the best of our knowledge, this important fact has never been explicitly reflected in the respective literature so far. Seen from another perspective, our PAC algorithm works on the basis of a DFA-based representation, which is only possible by the observation that the delay values can be mapped to a finite set of integer values. And indeed only this very last fact enables the preferred Perceptron algorithm to PAC-learn LTFs as arising from arbiter PUFs, c.f. [2]. We will elaborate more on this very interesting new direction in the journal version of this paper.

In [19] it has been demonstrated that the maximum delay deviation of each inverter used in the PUF chain is 9 picoseconds for both cases, i.e., direct and crossed paths on average for a Xilinx Virtex-5 FPGA. This delay difference is virtually in line with what has been observed in [17], where for 12 XC5VLX110 chips (Xilinx Virtex-5 family) the delay deviation is smaller than 10 picoseconds on all chips. Assuming $6\omega = 10$ ps, the maximum variation of delay at the end of the PUF chain consisting of n stages is $10\sqrt{n}$ ps. However, this value has to be divided by the precision of the arbiter to calculate the maximum value M , c.f. Sec. 3.1. For an absolute precise arbiter, the precision γ can be thought as being infinitesimal, i.e., $\gamma \rightarrow 1/\infty$, see [19]. Nevertheless, the precision of the arbiter is reported to be only in the range of 2.5 ps for a Xilinx Virtex-5 FPGA, c.f. [19]. As an example, under the assumption that $n = 128$,

$$M = \lceil 6\omega\sqrt{n}/\gamma \rceil = \lceil 10 \cdot \sqrt{128}/2.5 \rceil = 46.$$

Therefore, the size of the collapsed DFA would be only 282,752 states. This is far less than the size of $O(2^{128})$ states, which would be obtained without considering the limited variation of the delays.

Naturally, to make PAC-learning of a concrete PUF less effective, one is tempted to construct a PUF with a very large M . Theoretically, the maximum delay value M can be increased by enlarging the manufacturing deviations, and also using more precise arbiters. However, the deviation ω cannot be arbitrary large on real production chips. For instance, when increasing ω , a Field Programmable Gate Array (FPGA) cannot be utilized anymore as a general purpose device for other applications. Moreover, with regard to the higher cost, the PUF constructors cannot arbitrarily increase the precision of the arbiter. Due to these limitations, arbitrary large M 's can be excluded in practice for FPGAs and standard CMOS process devices.

In Section 3.2 we have stated that in practice it may rarely happen that $\alpha_{n,1} = \alpha_{n,2}$. This can be explained by the fact that this equation represents the possible metastable condition of the arbiter PUF, when the output of the arbiter is not persistent for a certain challenge c . Note that this metastability of the arbiter PUF must have been already solved by the PUF manufacturer. Moreover,

aiming at PAC-learning of an arbiter PUF under a DFA-based representation, we can also easily overcome this issue by applying two well-known strategies.

- The label of every chosen example, e.g., $f_{\text{PUF}}(c)$, (potentially having the metastability situation) will be stabilized by majority voting through several oracle calls on the same example (challenge).
- A problematic example resulting in different outputs at the arbiter can be simply discarded and substituted by another randomly chosen example.

We have proved that the number of CRPs required for PAC-learning is polynomial in n , M , $1/\varepsilon$ and $1/\delta$. This is contrary to what has been claimed and verified empirically in previous work, e.g., in [16]). For instance, according to an empirical formula presented in [25], the number of CRPs required for modeling an arbiter PUF with 128 stages for an $\varepsilon = 0.01$ is approximately 6450, although the upper-bound of the number of CRPs has not been calculated. The most critical issue which is not addressed in [25], is that the success probability of modeling an arbiter PUF for a given ε and a certain number of CRPs remains unknown. In other words, it is not ensured that for a given level of confidence, an arbiter PUF can be modeled for a given ε by collecting a polynomial-size sub-set of CRPs. Our proposed PAC-learning framework addresses this issue in a natural way.

The fact leading to the success of heuristic-based modeling attacks in general is that although an arbiter PUF with n stages allows for 2^n challenges, the number of delay values forming $\alpha_{n,1}$ and $\alpha_{n,2}$ is only linear in n [14]. Possible countermeasures against modeling attacks that have been introduced so far are based on adding non-linearity to arbiter PUFs, such as XOR or feed-forward arbiter PUFs [10]. It has been shown that these countermeasures are rather effective (see [25, 16]). Thus, the number of CRPs required for modeling is exponential in n in this case. We claim that although the non-linearity of an arbiter PUF may result in a more complex representation of that, our approach can be applied to break the security of the modified arbiter PUFs (XOR- and feed-forward PUFs) as well. For instance, in the case of XOR-PUFs, if the number of XORs is constant and independent of n , the DFA-representation that has been proposed by us can be modified to enable even a PAC-learning algorithm of an XOR-based arbiter PUF. Since the size of the modified DFA is still polynomial in n and M , the number of CRPs required for modeling is polynomial in n , M and $1/\varepsilon$ as well as $1/\delta$. The running time of algorithm remains polynomial in n , M and $1/\varepsilon$ as well as $1/\delta$. To sum all this up, due to the constant value of M for both, arbiter PUFs and its modified structures, it is possible to learn modified arbiter PUFs for given ε and δ . Furthermore, it is ensured that the PAC-learning algorithm runs in time polynomial in n , M , $1/\varepsilon$ and $1/\delta$.

6 Conclusion

In this paper, we have proved that arbiter PUFs are efficiently PAC-learnable under a DFA-based representation. As our DFA-based representation of the arbiter

PUF requires integer delay values, we have presented a statistical discretization and mapping process from the real values of the multiplexer delays to a set of integer values. Based on the limited variation in different delay values, we are able to construct a collapsed DFA, whose size is polynomial in the number of stages and maximum variation of delay values. It is proved that in contrast to other modeling attacks, our PAC-learning framework is provably able to learn the challenge-response behavior of the PUF for given accuracy and confidence levels. Furthermore, it has been shown that the maximum number of CRPs required for learning is polynomial in the number of stages, maximum deviation of delay values, as well as given levels of accuracy and confidence. Finally, we have evaluated the time complexity of our PAC-learning algorithm, which is polynomial in the length n of the arbiter PUF, the maximum variation of delays and levels of accuracy and confidence.

References

1. Angluin, D.: Learning Regular Sets from Queries and Counterexamples. *Information and computation* 75(2), 87–106 (1987)
2. Anthony, M.: Computational learning theory. Cambridge University Press (1997)
3. Armknecht, F., Maes, R., Sadeghi, A., Standaert, O.X., Wachsmann, C.: A Formalization of the Security Features of Physical Functions. In: *Security and Privacy (SP)*, 2011 IEEE Symposium on. pp. 397–412. IEEE (2011)
4. Delvaux, J., Verbauwheide, I.: Fault injection modeling attacks on 65nm arbiter and ro sum pufs via environmental changes. Tech. rep., *Cryptology ePrint Archive: Report 2013/619*, 2013, <https://eprint.iacr.org/2013/619> (2013)
5. Delvaux, J., Verbauwheide, I.: Side channel modeling attacks on 65nm arbiter pufs exploiting cmos device noise. In: *Hardware-Oriented Security and Trust (HOST)*, 2013 IEEE International Symposium on. pp. 137–142. IEEE (2013)
6. 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. ACM (2002)
7. Gassend, B., Lim, D., Clarke, D., Van Dijk, M., Devadas, S.: Identification and authentication of integrated circuits. *Concurrency and Computation: Practice and Experience* 16(11), 1077–1098 (2004)
8. Hammouri, G., Öztürk, E., Sunar, B.: A tamper-proof and lightweight authentication scheme. *Pervasive and mobile computing* 4(6), 807–818 (2008)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: *Automata Theory, Languages, and Computation*. International Edition 24 (2006)
10. Hospodar, G., Maes, R., Verbauwheide, I.: Machine Learning Attacks on 65nm Arbiter PUFs: Accurate modeling poses strict bounds on usability. In: *WIFS*. pp. 37–42 (2012)
11. Kearns, M.J., Vazirani, U.V.: *An Introduction to Computational Learning Theory*. MIT press (1994)
12. 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: *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*. pp. 176–179. IEEE (2004)

13. Lim, D., Lee, J.W., Gassend, B., Suh, G.E., Van Dijk, M., Devadas, S.: Extracting Secret Keys from Integrated Circuits. *Very Large Scale Integration (VLSI) Systems*, IEEE Transactions on 13(10), 1200–1205 (2005)
14. Maes, R.: *Physically Unclonable Functions: Constructions, Properties and Applications*. Springer Berlin Heidelberg (2013)
15. Maes, R., Verbauwhede, I.: Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. In: *Towards Hardware-Intrinsic Security*, pp. 3–37. Springer (2010)
16. Mahmoud, A., Rührmair, U., Majzoobi, M., Koushanfar, F.: Combined Modeling and Side Channel Attacks on Strong PUFs. Tech. rep., Cryptology ePrint Archive: Report 2013/632, 2013, <https://eprint.iacr.org/2013/632> (2013)
17. Majzoobi, M., Dyer, E., Elnably, A., Koushanfar, F.: Rapid FPGA Delay Characterization Using Clock Synthesis and Sparse Sampling. In: *Test Conference (ITC)*, 2010 IEEE International. pp. 1–10. IEEE (2010)
18. Majzoobi, M., Koushanfar, F., Devadas, S.: FPGA PUF Using Programmable Delay Lines. In: *Information Forensics and Security (WIFS)*, 2010 IEEE International Workshop on. pp. 1–6. IEEE (2010)
19. Majzoobi, M., Koushanfar, F., Devadas, S.: FPGA PUF Using Programmable Delay Lines. In: *Information Forensics and Security (WIFS)*, 2010 IEEE International Workshop on. pp. 1–6. IEEE (2010)
20. Majzoobi, M., Koushanfar, F., Potkonjak, M.: Techniques for Design and Implementation of Secure Reconfigurable PUFs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 2 (2009)
21. Papoulis, A., Pillai, S.U.: *Probability, Random Variables, and Stochastic Processes*. Tata McGraw-Hill Education (2002)
22. Parekh, R., Nichitru, C., Honavar, V.: A Polynomial Time Incremental Algorithm for Learning DFA. In: *Grammatical Inference*, pp. 37–49. Springer (1998)
23. Parusiński, M., Shariati, S., Kamel, D., Xavier-Standaert, F.: Strong PUFs and their (Physical) Unpredictability: A Case Study with Power PUFs. In: *Proceedings of the Workshop on Embedded Systems Security*. p. 5. ACM (2013)
24. Rührmair, 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 (2013)
25. 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. ACM (2010)
26. Rührmair, U., Sölter, J., Sehnke, F.: On the Foundations of Physical Unclonable Functions. *IACR Cryptology ePrint Archive* 2009, 277 (2009)
27. Sadeghi, A.R., Visconti, I., Wachsmann, C.: *Enhancing RFID Security and Privacy by Physically Unclonable Functions*. Springer (2010)
28. Škorić, B., Tuyls, P., Oprey, W.: Robust Key Extraction from Physical Unclonable Functions. In: *Applied Cryptography and Network Security*. pp. 407–422. Springer (2005)
29. 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. ACM (2007)
30. Tajik, S., Dietz, E., Frohmann, S., Seifert, J.P., Nedospasov, D., Helfmeier, C., Boit, C., Dittrich, H.: Physical Characterization of Arbiter PUFs. In: *Cryptographic Hardware and Embedded Systems—CHES 2014*. Springer (2014)
31. Tuyls, P., Batina, L.: RFID-tags for Anti-Counterfeiting. In: *Topics in Cryptology—CT-RSA 2006*, pp. 115–131. Springer (2006)