Kalpa Publications in Computing

Volume 11, 2019, Pages 33–48

Proceedings of 8th International Workshop on Security Proofs for Embedded Systems

Rock'n'roll PUFs: Crafting Provably Secure PUFs from Less Secure Ones

Fatemeh Ganji[1], Shahin Tajik[1], Pascal Stauss[2], Jean-Pierre Seifert[2], Domenic Forte[1], and Mark Tehranipoor[1]

[1]Florida Institute for Cybersecurity Research, University of Florida, USA
[2]Security in Telecommunications Group, Technische Universität Berlin, Germany
{fganji,stajik,dforte,tehranipoor}@ufl.edu
{pstauss, jpseifert}@sect.tu-berlin.de

Abstract

The era of PUFs has been characterized by the efforts put into research and the development of PUFs that are resilient against attacks, in particular, machine learning (ML) attacks. Due to the lack of systematic and provable methods for this purpose, we have witnessed the ever-continuing competition between PUF designers/ manufacturers, cryptanalysts, and of course, adversaries that maliciously break the security of PUFs. This is despite a series of acknowledged principles developed in cryptography and complexity theory, under the umbrella term "hardness amplification". This paper aims at narrowing the gap between these studies and hardware security, specifically for applications in the domain of PUFs. To this end, this paper provides an example of *somewhat* hard PUFs and demonstrates how to build a *strongly* secure construction out of these considerably weaker primitives. Our theoretical findings are discussed in an exhaustive manner and supported by the silicon results captured from real-world PUFs.

Keywords: Hardness Amplification, Machine Learning, Physically Unclonable Functions

1 Introduction

After the introduction of the first strong physically unclonable function (PUF), it became soon evident that it is vulnerable to machine learning (ML) attacks [17]. In this case, an adversary can intercept the transmission of a subset of PUF challenge-response pairs (CRPs) between a prover and a verifier, and run an ML algorithm on the gathered CRPs to create a model of the PUF. Several countermeasures, from structural to protocol level, have been proposed to increase the security of PUFs against ML attacks. However, one of the main questions remains unanswered: is it possible to create a significantly harder PUF from a PUF, which is moderately resilient to ML attacks? In ML theory as well as cryptography, this question has been already addressed through hardness amplification approaches, mainly by the Yao's XOR Lemma. This lemma states that if several instances of a somewhat-hard functions are XORed together, the resulting function is harder to compute. One can observe a close relationship between the concept of XOR PUFs and Yao's lemma, although it has not even

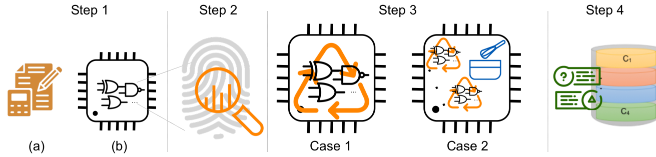

Figure 1: A systematic hardness amplification methodology. In Step 1, the designer defines the hardness level and number of CRPs in each round of rolling, and designs the PUF (marked by (a) and (b), respectively). The influential stages are computed in Step 2. By rolling these stages, a somewhat hard PUF can be built, as shown in Step 3 (Case 1). If the accuracy of this PUF is still unsatisfactory, some instances of that can be combined to improve the hardness (Case 2). Finally, in Step 4, the designer limits the access of an attacker to the CRPs by allowing only a pre-defined number of CRPs to be exchanged in each round of rolling.

implicitly discussed in the PUF-related literature, to the best of our knowledge. The application of the XOR function in the context of PUFs has its roots in the problem of how to obfuscate the outputs of a PUF [27]. Besides, this function has been employed to enhance the performance characteristics of PUFs in terms of, e.g., reliability, uniqueness, etc. (for a prime example, see, [25]). Nevertheless, when it comes to resilience to ML attack, it has been shown that XORing can be only marginally effective since empirical and provable attacks are launched to compromise their security [4, 11, 24]. Therefore, it is as yet unclear how and to what extent the *hardness* of a PUF, i.e., being mildly resilient against ML attacks, can be amplified. To address this, our framework takes some steps towards presenting a provably secure PUF, whose hardness is amplified as commonly practiced in cryptography and ML theory.

Our contribution. The main contribution of this work is proposing a framework for the hardness amplification of PUFs. In this regard, we first discuss an *example* of how a somewhat hard PUF can be designed in practice. In doing so, we propose to add *uncertainty* to the PUF architecture by rolling a PUF to make ML attacks fail. We exploit the *partial* reconfigurability of modern FPGAs to swap only a few stages of a PUF, while the other parts of the PUF remain intact (for a literature review on this, see Sect. 3). To this end, we develop a systematic methodology through the concept of influential stages [10] to identify which PUF stages should be rolled to flip the responses of the PUF to given challenges. These stages are then reconfigured.

Although the above approach makes the PUF more resilient to ML attacks, it still requires drastic practical measures to be taken to achieve the ultimate level of security, where the attacker cannot do considerably better than flipping a coin for the response of the PUF. Hence, we shift our focus to known approaches for hardness amplification acknowledged in the ML theory. In this context, we demonstrate that the *Tribes functions* outperform the XOR function in PUF scenarios in terms of not only resiliency to ML attacks, but also being close to balanced (i.e., unbiased). Furthermore, we introduce a new metric, namely, the expected bias, which indicates how hard a combined PUF can be. Last but not least, to support our theoretical findings, we conduct comprehensive experiments – followed by an extensive discussion – on rolling Arbiter PUFs implemented on the Xilinx Artix 7 FPGAs. It should be noted, that we have selected Arbiter PUFs for our experiments only since there is a verified systematic way to implement it on an FPGA [19]. Hence, the proposed framework in this paper is not limited to one specific PUF family, and thus, it could be applied to other strong PUF implementations as well.

A brief overview of the proposed scheme: Although the hardness amplification theory, forming the base for our framework, might have been known in the ML community, this paper presents the first formalization of a secure PUF design in the context of this theory. More

importantly, as we demonstrate, without a profound support from the hardware platform embodying a PUF, the concept of hardness amplification cannot be realized in practice. To give a better understanding of this, Fig. 1 illustrates the steps of our methodology described below. **Step 1** concerns the design of a PUF, marked with (a) in Fig. 1. Additionally, the designer can define the level of hardness (Step 1(b)). Afterward, in **Step 2**, the set of PUF stages that should be rolled is determined in a systematic manner, namely by computing the influential stages that determine the responses of the PUF to given challenges. **Step 3** encompasses two cases: designing a somewhat hard PUF and combining such PUFs to achieve the ultimate hardness as desired by the designer in Step 1. We show that to obtain a somewhat hard PUF, it is inevitable to enjoy the practical advantages offered, e.g., by the rolling strategies. Moreover, we prove that the Tribes function outperform the XOR function to combine somewhat hard PUFs. Finally, in **Step 4**, to help with this entire process, we limit the number of CRPs accessible to an adversary in each rolling round. For this purpose, we adopt a simple but effective strategy: in each round of rolling, the number of CRPs does not exceed the lower bound established for provable ML algorithms in general. This lower bound can be also calculated upfront, in Step 1, by the designer.

2 Notation and Preliminaries

2.1 Boolean Functions as Representations of PUFs

Generally speaking, PUFs can be defined by physical mappings from the given *challenges* to the respective *responses*. PUFs exhibiting an exponentially large set of CRPs are called strong PUFs. The physical mappings underlying the design of *digital intrinsic* PUFs are characterized by inherent silicon imperfections of the hardware, on which the PUF is implemented. Here we consider merely unclonability among several security properties of PUFs. Let the mapping $f_{\mathrm{PUF}}: \mathcal{C} \rightarrow \mathcal{Y}$, where $f_{\mathrm{PUF}}(c) = y$, describes a PUF. Ideally, for a given PUF f_{PUF}, unclonability reflects the fact that creating a clone, i.e., a (physical) mapping $g_{\mathrm{PUF}} \neq f_{\mathrm{PUF}}$, is virtually infeasible, where the challenge-response behavior of g_{PUF} is *similar* to f_{PUF} [3]. We stress that our work does not cover the topics of formalization of the PUFs (for more details on this, see, e.g., [3]). Similar to the most relevant studies on PUFs, we represent PUFs as Boolean functions over the finite field: $f_{\mathrm{PUF}} = \{0,1\}^n \rightarrow \{0,1\}$.

In this paper, when we occasionally need a representation of Boolean functions, we stick to Linear Threshold functions (LTFs) and k-junta functions. A k-junta is a Boolean function, whose output is determined merely by an unknown set of k variables. In order to define an LTF, the encoding scheme $\chi(0_{\mathbb{F}_2}) := +1$, and $\chi(1_{\mathbb{F}_2}) := -1$ should be considered, based upon which the Boolean function f can be defined as $f : \{-1,+1\}^n \rightarrow \{-1,+1\}$. Now, a Boolean function is called a linear threshold function (LTF), if there are coefficients $\omega_1, \omega_2, \cdots, \omega_n \in \mathbb{R}$ and $\theta \in \mathbb{R}$ such that $f(c) = \operatorname{sgn}((\sum_{i=1}^{n} \omega_i c_i) - \theta)$. Here we assume, without loss of generality, that $\sum_{i=1}^{n} \omega_i c_i \neq \theta$ for every $c \in C_n$.

Tribes Functions [22]: A Tribes function on k Boolean variables is defined as follows. $C(x_1, \cdots, x_k) := (x_1 \wedge \cdots \wedge x_b) \vee \cdots \vee (x_{k-b+1} \wedge \cdots \wedge x_k)$, where $\wedge$ and $\vee$ denote the logical AND and OR functions, respectively. Moreover, b is the largest integer fulfilling the inequality $(1 - 2^{-b})^{k/b} \geq 1/2$; hence, $b = O(\log k)$.

2.2 Probably Approximately Correct Learning Model

Here, we stick to an extension of the PAC model, beneficial for our security analysis, where the examples are informally drawn from the instance space $C_n = \{0,1\}^n$. For the target concept

class F_n and the hypothesis class H_n, the hypothesis $h \in H_n$ being a Boolean function over C_n is an ε-approximator for $f \in F_n$, if $\Pr_{c \in_U C_n}[f(c) = h(c)] \geq 1 - \varepsilon$, where the index U shows that the example are drawn with regard to the uniform distribution. Regarding the definition of the hypothesis class, two cases can be differentiated. In the first case, the algorithm is forced to deliver a given, pre-defined hypothesis class that is subset of the target class $F = \cup_{n \geq 1} F_n$, while in the second case, the algorithm is allowed to deliver the hypothesis $h \in H_n$ so that $H \not\subset F$, and consequently, $h \notin F$. The first case is known as *proper* learning, whereas the latter is referred to as *improper* learning. Now we can define a uniform PAC learning algorithm as follows.

Definition 1. *The algorithm* $\mathtt{A}$ *learning the target concept class* F *is given a polynomial number of labeled examples drawn uniformly from* C_n*. The algorithm then returns an* ε*-approximator for* f*, the hypothesis* h*, with probability at least* $1 - \delta$*. For any* $n \geq 1$*, any* $0 < \varepsilon, \delta < 1$*, and any* $f \in F_n$*, the running time of* $\mathtt{A}$ *is* $poly(n, 1/\varepsilon, VC_{dim}(F), 1/\delta)$*, where* $poly(\cdot)$ *denotes a polynomial function.*

In the above definition, the Vapnik-Chervonenkis (VC) dimension $\mathrm{VC}_{\mathrm{dim}}(F)$ offers the measure of complexity of the target concept class. In this regard, generally speaking, for a PAC learning algorithm it is possible to establish a lower bound on the number of examples needed to obtain the ε-approximator for $f \in F$ with the probability at least $1 - \delta$ [5]: $N = \Omega\left(\frac{1}{\varepsilon}\log\frac{1}{\delta} + VC_{dim}(F)\right)$.

3 Designing a Somewhat Hard PUF

It is evident that in order to apply the concept of hardness amplification, one has to first come up with a somewhat hard function, i.e., a PUF in our case. The harder the PUF that we begin with, the resulting, final PUF would be more cost-efficient (in terms of the number of PUFs to be combined) and harder-to-learn. In other word, for our purpose, a PUF family that can be learned with a high accuracy is not useful. This section presents an example of how somewhat hard PUFs can be constructed in practice.

The notion of "rolled" PUFs: Among all potential solutions concerning the resilience of PUFs against ML attacks, a class of countermeasures relies on *rolling* the PUF during authentication. Similar to the concept of key rolling, where the key of a cryptographic algorithm is regularly updated during encryption/decryption to limit the leakage of side-channel information, a PUF instance can be rolled and swapped with new PUFs to limit the amount of exposed CRPs from each specific PUF instance. As a result, the attacker cannot obtain enough CRPs for each individual rolled PUF to create accurate models for them, see, e.g., [28,30]. Although applied on a trial-and-error basis and in a *blind* fashion, this approach has been shown to be helpful. This can be explained by a fact that all the above mentioned rolling strategies aim at adding *noise* to the PUF architecture. A great deal of attention should be paid to differentiate this noise from the phenomenon commonly referred to under the general term "noise".

The noise associated with the rolling can be thought of as a *hidden variable* [18]. It has been shown that the hidden variables, namely a manufacturing process variable and a noise variable, account for the underlying physical, random processes in a PUF stage. To relate this model to the noise models known in ML theory, it has been shown that the hidden variable in a PUF stage refers to the "attribute" noise in the ML related literature [12]. More formally, similar to the models presented in [12,18], consider a (silicon) PUF stage, e.g., i^{th} stage, with the process variations X_i $(1 \leq i \leq n)$ and the noise varying during each evaluation $N_i \sim \mathcal{N}(\mu_N, \sigma_N^2)$. X_i follows a Gaussian distribution with the mean value μ reported by manufacturers as the nominal

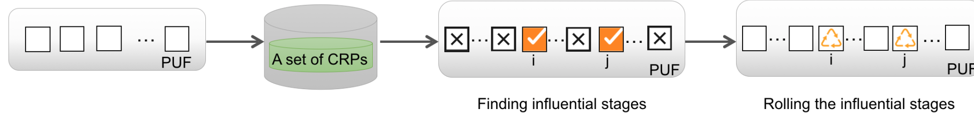

Figure 2: From a conventional PUF to its rolled version. First, the designer should collect a sub-set of CRPs and run a test (e.g., the test in [21]) to determine the influential stages of the PUF, i.e., the stages associated with the Boolean variable determining the output of the function f_{PUF}. To obtain the rolled PUF, various strategies can be employed (see Sect. 6).

value, and the standard deviation σ_i, which is the result of the process variations [20]. Clearly, the total impact of hidden variables on a stage can be written as $Z_i = X_i + N_i$ $(1 \leq i \leq n)$ with Z_i being a Gaussian random variable. By applying the challenge bits $c_i = 1$ and $c_i = 0$, we obtain $z_{i,1} = x_{i,1} + n_{i,1}$ and $z_{i,0} = x_{i,0} + n_{i,0}$, respectively.

When reconfiguring a stage of the PUF, irrespective of the reconfiguration method, the realizations $x_{i,1}$ and $x_{i,0}$ are changed to $x'_{i,1}$ and $x'_{i,0}$. Although the latter two realizations can be very close to those in the first setting, due to the manufacturing process variations, they are not identical. Note that $x'_{i,1}$ and $x'_{i,0}$ follow the same distribution as $x_{i,1}$ and $x_{i,0}$. Following the definitions presented in [12, 18], the realizations of the random variable Z_i can be rewritten to obtain $z_{i,0} = x'_{i,0} + n'_{i,0}$, where $n'_{i,0} = x_{i,0} - x'_{i,0} + n_{i,0}$. In order to relate the random variable Z_i to the concept of attribute noise, we should determine the conditions under which $z_{i,0} = x_{i,1}$, cf. [12]. In other words, in the above circumstance, it is impossible to decide which challenge bit ("0" or "1") is applied to the PUF. The same conditions can be met for the reconfigured PUF stage, when $x'_{i,0} = x_{i,1} - n'_{i,0}$, and $x'_{i,1} = x_{i,0} - n'_{i,0}$, it is not straightforward to say which configuration/ challenge bit is used. To sum up, here we demonstrate that reconfiguration of a PUF stage can be translated to the attribute noise in the ML domain.

Rolled PUFs as somewhat resilient PUFs: When coming under ML attacks, PUFs enhanced by adding attribute noise can resist better since to obtain a sufficiently accurate model of the PUF with a high probability, a larger number of CRPs is required by ML algorithms. This number of CRPs is polynomial in the level of the attribute noise, and hence, increasing in this level results in achieving better resiliency against ML attacks (see Corollary 4 in [12]). Therefore, if a framework can be established to increase the level of attribute noise beyond an upper bound, the effectiveness of an ML attack against the PUF can be impaired. To this end, our framework suggests that (some) stages of the PUF are *rolled* in several rounds to obtain a more ML attack-resistant primitive. Clearly, a *blind* partial reconfiguration cannot automatically assure making a change in the challenge-response behavior of the PUF. More precisely, adding the attribute noise can have a significant impact on the outputs of the function, but only if influential challenge bits are affected by the attribute noise; otherwise, the output of the function is not flipped [2]. Therefore, before rolling a PUF, it is crucial to determine the influential stages, for instance, by applying an approach proposed in [10], see Fig. 2.

The importance of introducing rounds into our scheme: This issue is two-fold. (1) If the rolled stages are the same in each round, it may be still possible to learn the PUF by mounting the Fourier-based attack [12]. This is due to the fact that by reconfiguring the PUF stages, we introduce the attribute noise into our PUF, which can only slightly impair the effectiveness of the above attack. (2) Different sets of CRPs should be used in the rounds. This is in line with other cryptographic schemes aiming at preventing replay attacks as well as the concept of controlled PUFs proposed in the literature, e.g., [13],(see Sect. 7 for discussion on the difference between controlled PUFs and our scheme). The goal is to stop an attacker attempting to collect

noisy CRPs in each round of rolling, the attacker should not have access to a set of CRPs needed to build a sufficiently accurate model of the challenge-response behavior of the PUF. To address this issue, a lower bound on the number of examples required by the learning algorithm can be established (see Sect. 2.2). Afterward, the number of CRPs used in each round should not exceed the lower bound.

As an example, consider Arbiter PUFs represented by LTFs. As for LTFs with n Boolean variables the VC dimension is $n+1$, it is easy to calculate the lower bound as $N = \Omega\left(1/\varepsilon \log(1/\delta) + n + 1\right)$ (see Sect.2.2). Now by setting $\varepsilon = 0.51$, we ensure that the attacker cannot do considerably better than flipping a coin to determine the response of the PUF to a given challenge. As an example, for a *noiseless* 64-bit Arbiter PUF, the minimum number of CRPs required to achieve our learning goal is approximately 75, when $\delta = 0.01$. This bound holds in general in the sense that regardless of the algorithm applied to learn a target concept, the above bound can be established. More interestingly, the bound discussed above is achieved for the noiseless setting. In other words, in the presence of noise, the accuracy of the model is less than ε for the same number of CRPs and the confidence level (δ).

4 Hardness Amplification for PUFs

Even though the design of the somewhat hard PUF discussed in Sect. 3 is further supported by theoretical analysis, the desired hardness may not be achieved easily. This is due to the fact that such an approach is still dependent on the practical aspects of the design, such as manufacturing variations, which are not entirely under our control. Therefore, we claim that even when employing a systematic approach, as considered in the first part of our paper, a paradigm shift from practical methods to theoretical and provable ones should be emphasized.

As discussed in Sect. 1, we are interested in building a strongly secure construction out of a weakly secure construction of a cryptographic primitive. To address this, Yao's XOR lemma has been established, which states: (informally) if a Boolean function f is weakly unpredictable (i.e., learnable) within some complexity bound, for sufficiently large k, the function $F(x_1, ..., x_k) := \bigoplus_{i=1}^{k} f(x_i)$ is virtually unpredictable within the respective complexity bound. This lemma has been further strengthened to ensure that the final function F is close-to-balanced, and in particular, remaining close-to-balanced even in the presence of the attribute noise, see Theorem 1 [22].

Theorem 1. [22] *Let the function $f : \{0,1\}^n \rightarrow \{0,1\}$ denotes a Boolean function in NP that is $1/poly(n)$-hard to be computed. There exist a function $F(x_1, ..., x_k) := C(f(x_1), \cdots, f(x_k))$ that is $1/2 + n^{-1/3+\varepsilon}$ hard, where C is a combination function.*

Here we do not discuss the proof of this theorem, but the properties of the function C is of great importance to us. As proven in [22], the function C should be polynomial-time computable and monotone, i.e., given two inputs $x \geq y$ in the partial order on $\{0,1\}^n$, we have $C(x) \geq C(y)$. Moreover, the combination function C should exhibit low *expected bias*, known as a metric indicating the hardness of the final, combined function. Informally, the expected bias is a measure of the average bias of a function in the presence of attribute noise. To introduce a combination function with the above properties, the "Tribes" function, has been used in the proof of Theorem 1. The inherent feature of the Tribes functions making them suitable for the hardness amplification is not only their monotonicity, but also remaining close-to-balanced even in the presence of the attribute noise.

Now we can eventually shift our focus to hardness amplification for PUFs. We argue that the hardness amplification following the Theorem 1 is much stronger than what can be achieved

by XORing the PUFs. In particular, the new combination functions, i.e., Tribes functions, provides an inherently distinctive characteristic that makes them attractive in PUF scenarios: (1) on the one hand, Tribes functions are sufficiently noise sensitive to ensure a close-to-random behavior of the combined function, and (2) on the other hand, they are nearly balanced and preserving this quality even when being subject to the attribute noise. The following theorem concludes this section by stating how the hardness of PUFs can be amplified.

Theorem 2. *Consider a PUF $f_{PUF} : \{0,1\}^n \to \{0,1\}$ that is given N pairs of CRPs chosen uniformly and an adversary using these N pairs to come up with an ε_N-approximator for f_{PUF} with probability at least $1-\delta_N$. There exist a Tribe function C applied to k instances of f_{PUF}, i.e., $C(f_{PUF}(x_1), \cdots, f_{PUF}(x_k))$ to obtain a $1/2 + n^{-1/3+\varepsilon}$-hard function.*

Proof: First set the desired confidence level δ, and N being the minimum number of CRPs that a polynomial-time algorithm requires to deliver an ε_N-approximator of the function f_{PUF} with the desired probability $1-\delta$. Clearly, according to the PAC model, ε_N is a function of δ, N, and n. More precisely, the achievable accuracy level is indeed $1/poly(n)$ (see Sect. 2.2). Note that with this number of CRPs no polynomial-time algorithm can achieve better than an ε_N-approximation of f_{PUF}. The proof is completed by taking Theorem 1 and setting $k = O(n^{2/3})$. ■

How many CRPs are required in each round of rolling Tribes-combined PUFs? We should establish the lower bound on the number of CRPs needed to learn a PUF with the accuracy of $1-\delta$, and then the number of CRPs in each round should be less than that. We begin with the calculation of the VC dimension of the combined function $C(f_{\mathrm{PUF}}(x_1), \cdots, f_{\mathrm{PUF}}(x_k))$, where $C(\cdot)$ is the Tribes function. For the representation of the target concept, f_{PUF}, we stick to the LTF functions. It is known that for finite concept classes, $\mathrm{VC}_{\dim}(C \otimes F) = O(kd + kd\log(kd))$, where $\otimes$ denotes the composition operator and $d = \mathrm{VC}_{\dim}(F)$. When the inputs to this Tribes function are LTFs with $\mathrm{VC}_{\dim}(F) = n+1$, we obtain $\mathrm{VC}_{\dim}(C(f_{\mathrm{PUF}}(x_1), \cdots, f_{\mathrm{PUF}}(x_k))) = O(k(n+1)(1+\log(kn+k)))$. Substituting this in the general lower bound (see Sect. 2.2) yields the minimum number of CRPs required to learn our combined function. Note that the ultimate hardness of our scheme depends not only on the hardness of the PUF chains, but also on the expected bias of the final combined function [22]. As the expected bias of the Tribes functions is close to 1/2, in contrast to the XOR function, the hardness of the combined function can be guaranteed.

5 PUF Implementation

One approach to roll a PUF circuit on an ASIC or FPGA is to have multiple PUF instances in parallel, where they are selected by a multiplexer [26]. Another approach is to physically swap a PUF circuit partially or entirely, which is only feasible by using the dynamic reconfigurability feature of mainstream FPGAs [28]. The first introduced rolled PUF [28] has utilized the reconfiguration capability of an FPGA to reconfigure and move the entire PUF circuit on the FPGA. However, inside an FPGA the resources are shared among several IPs and free movement of the PUF might not be feasible. Moreover, implementing a strong PUF, such as Arbiter PUF, on an FPGA requires precise and symmetric routing constraints, which is not achievable by random reconfigurations. Even a complete symmetric design does not necessarily guarantee that the PUF works properly, i.e., without a far too high level of bias and acceptable inter/intra distances. Hence, it is more desirable to partially reconfigure the PUF and swap only a few stages of it. More importantly, partial reconfiguration provides support for our theoretical

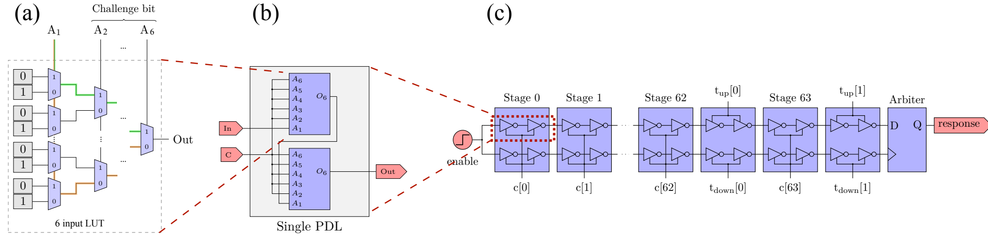

Figure 3: (a) Illustration of the propagation delay in a 6-input LUT based on the state of the challenge bit. (b) Composition of the PDL from two 6-input LUTs which operate as inverters. (c) Schematic structure of a PUF using PDLs as stages.

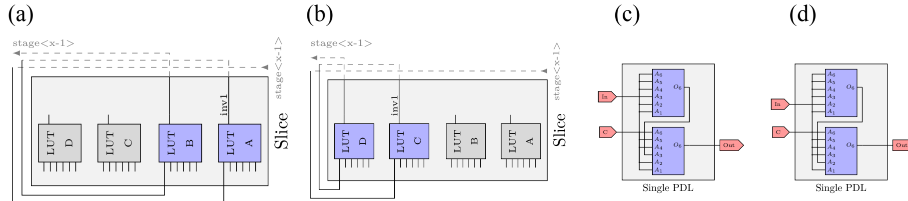

Figure 4: Examples of rolling the PUF stages with different LUT combinations in a slice and different physical LUT inputs: (a) using LUTs A and B for an unrolled stage, (b) Swapping both LUTs to LUTs C and D. (c,d) changing the input from A_1 to A_2 and A_3, respectively.

setting, namely, only the influential stages can be reconfigured in each round of the rolling. Here, we describe the setup used to realize the approach discussed in Sect. 3.

FPGA implementation of Arbiter PUFs: We deploy a Digilent Arty development board, which contains a Xilinx Artix-7 (XC7A35T) FPGA manufactured with 28 nm technology. Among different PUF candidates, we selected Arbiter PUFs for our experiments since there is a verified systematic way to implement them on FPGAs [19]. Due to delay imbalances of classical multiplexer-based Arbiter PUF on FPGAs, we implemented the stages by two independent chains using so-called programmable delay line (PDL) as in [19], see Fig. 3. The PDL takes advantage of the internal behavior of a LUT and consists of two LUTs in the implementation, where each LUT behaves as an inverter, see Fig. 3a. The signal applied to A_1 takes different distances, depending on the assignment of the inputs of $A_2 - A_6$. If $A_2A_3A_4A_5A_6 = 11111$, the highlighted path in green is selected, see Fig. 3a; Otherwise, if $A_2A_3A_4A_5A_6 = 00000$, the longer red path is chosen. If we take 6-input LUTs of our DUT into account, input A_1 can be used as inverter input, and the remaining inputs $A_2 - A_6$ serve as our interconnected challenge bit. By implementing a second inverter connected in series, the caused signal inversion is reversed, see Fig. 3b. A complete stage of the PUF can be realized as a module from two such individual PDL module instances, each with the same challenge bit. Our design consists of a 64-bit Arbiter PUF, where 64 of these modules are chained in series, see Fig. 3c. Additionally, there is a certain number of tuning stages [19], consisting of two PDL instances similar to the PUF stages. However, they are realized with two independent selector bits, which compensate for the bias between the two routes. At the end of the chain, a D-flip-flop serves as an arbiter, where data/clock input is connected to the upper/lower path. The response of the PUF is zero/one if the total delay through all stages of the upper signal path is greater/less than that of the lower.

Reconfig. stages	ε	# Influential stages	Influential stages
9, 22	0.0024	7	32,5,17,47,3,20,7
9, 17	0.0024	16	47,18,32,20,7,5,22,35,6, 54,31,3,53,10,50,4
5, 17	0.0024	2	9,22

Table 1: Influential stages within $n = 64$ stages. In this experiment, $\delta = 0.99$, the inside-LUT configuration is CD, and a set of stages in the first column is reconfigured. For the configurations *not shown* here, no influential stage is found.

5.1 Partial Reconfiguration of PUFs

At the first stage, we implement a reference Arbiter PUF. We take into account two different approaches for the reconfiguration: 1. changing the configuration of the LUTs, and 2. choosing different ports for the input signal. These modification options are based in part on the ideas presented in [14], where the reconfiguration of a ring oscillator PUF has been discussed. Considering the first approach, there are four LUTs A, B, C and D inside each slice of our FPGA, where in the initial unrolled stage, the LUTs A and B are deployed as PDLs. Since there are six combinations per stage (see Fig. 4) after taking into account the combination already used for the unrolled PUF, five combinations remain to construct a rolling with the selection of the two necessary LUTs. In the second approach, the propagation path can be changed within a stage for each PDL by selecting a different physical LUT input, see Fig. 4.

5.2 Measurements

For each experiment, we collect CRPs from the reference PUF and the rolled one obtained by applying one of the approaches mentioned in Sect. 5.1. The tuning stages are adjusted in a way that a ratio of nearly 50% of zeros and ones is obtained. Note that inter slice routing has the main effect on the bias of the PUF, and thus, retuning of tuning stages is not needed if reconfiguration is carried out inside a slice. Due to very little delay differences between two chains, the arbiter can sample a meta-stable signal. Thus, for each given challenge, we measured the responses 19 times, as found effective experimentally, and applied majority voting. Afterward, if the percentage of responses to a challenge equal the majority value is less than 75%, the CRP is marked as noisy and sorted out. From each PUF, we collect 640000 CRPs from that 89284 CRPs are removed due to the noisy behavior. Note that for each experiment we randomly and uniformly choose solely 113729 CRPs, depending on the value of ε and δ. The reason behind collecting more CRPs is to run the experiment repeatedly to obtain statistically relevant results. The CRPs are collected in an offline fashion, and then, analyzed by applying in-house developed algorithm written in MATLAB [16].

6 Results

6.1 Determining the Influential PUF Stages

As the first step, by applying a property testing algorithm, namely a k-junta testing algorithm [21], we determine the influential stages of the PUF. The rationale behind a k-junta testing algorithm is that giving black-box query access to an unknown function f, the algorithm determines whether f is ε-close to k-junta functions or it is "far" from any such function. The k-junta testing algorithm can also determine the influential stages. The number of queries made by the algorithm, i.e., CRPs in our case, is $poly(k)/\varepsilon$ [9]. In this regard, in Matlab [16] we implement the k-junta testing algorithm proposed in [21][1]. We run the algorithm against

[1] The algorithms are available under https://www.trust-hub.org/software

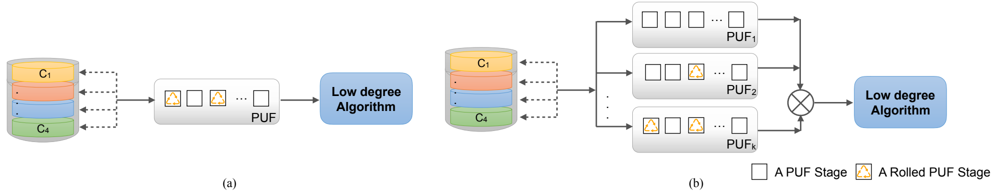

Figure 5: The schematic illustrating how our scheme is used in different rounds of rolling. The combination function shown in this scheme is either the XOR or the Tribes functions. The response of the PUF is given to the low degree algorithm to evaluate the hardness of the PUF.

our unrolled Arbiter PUF, i.e., a PUF with an inside-LUT configuration AB. With regard to the requirement enforced by the spectral properties of a Boolean function, e.g., Parseval's identity cf. [21], we compute an interval, within which ε can lie. The mean of the values lying within this interval is chosen as the default value of ε that is, $\varepsilon = 0.0024$. Moreover, we repeat each experiment 5 times and take the maximum number of influential stages. With regard to our results, in line with the theoretical limits on the number of influential stages in an LTF (see [1,12]), the unrolled arbiter PUF depends solely on 4 influential stages, namely, the stages 5,9,17, and 22.

In the second step of our experiments, we change the inside-LUT configuration of these stages one-by-one to AC, AD, BC, BD, and CD. In a nutshell, the following conclusions drawn from all of our experiments are worth noting here (for the complete set of results, see [1]). Among all inside-LUT reconfigurations, we observe that changing the configuration from AB to CD is the most effective strategy. Moreover, for some stages, there are some ineffective inside-LUT rolling strategies, e.g., changing the configuration from AB to the AD in the stage (22) and the AC in the stage (5) do not impair the effect of these influential bits (hereafter, the numbers inside the parentheses indicate the rolled stages). Secondly, the natural question, which may arise here, is that whether by collecting more CRPs, the attacker can find the influential stages of the rolled PUF. This can be explained by the fact that the influence of the stages is very well reduced so that increasing the precision of the algorithm results in finding none or only some of the influential stages. Finally, for the same value of ε, inside-LUT reconfigurations are more effective than the input-port selection strategy.

In the next step of our experiments, we observe that when rolling the PUF by changing the configuration of a set of stages (i.e., more than one stage at the same time), the traces of the remaining influential stages can be seen in some experiments, see Table 1. However, for some rolling strategies, the influential stages cannot be determined, even after increasing the number of available CRPs, e.g., after rolling all four influential stages. To sum up this section, we stress that in order to obtain a somewhat hard PUF through rolling, one has to go step-by-step and identify the best strategy to roll the PUF: a strategy that maximizes the difference between the sets of influential bits for a PUF and its rolled version.

6.2 Learning Rolled PUFs

Here, we examine if the rolled PUFs are still vulnerable to machine learning attacks, in particular, applying the low degree algorithm. The reason for this is two-fold. First, as discussed in Sect. 3, the reconfiguration of a PUF stage can be translated to an increase in the attribute noise level. Since it has been proven that even in the presence of this type of noise, the low degree algorithm can be applied [6], one can deduce that the security of rolled PUFs can be broken by mounting the low degree algorithm. Secondly, the low degree algorithm is categorized as an improper learning algorithm, i.e., without limit on the hypothesis class, hence, the infeasibility

Stage	# CRPs in training set	Accuracy
Unrolled (AB)	125	94.26
(9)		92.96
(17,22)		92.13
(5,9,17,22)		95.21

Table 2: Results for learning Arbiter PUFs with different number of rolled stages. The number of CRPs in the test set is 100000, and the inter-LUT configuration is CD.

of applying the low degree algorithm provides a stronger security assurance.

To employ the low degree algorithm, however, the attribute noise level should be known/ approximated. As in practice, this parameter is not under the designer's control, the best estimation of it, i.e., $\varepsilon = 0.05$, has been used in our experiments. The number of CRPs used to come up with an approximator of the Boolean function f_{PUF} is aggressively reduced to verify the lower bound on the number of CRPs discussed in Sect. 3. Regarding this, the number of CRPs in the training set is 125, whereas in the test set we have 100000 CRPs. Table 2 shows the accuracy of the model delivered by the low degree algorithm for only a set of PUFs, with different numbers of rolled stages, whose models have the least accuracy. Note that as we are interested in amplifying the hardness of PUFs, the minimum accuracy is important to our approach (for a complete set of results, see [1]). Nonetheless, we emphasize that these somewhat hard PUFs can be modeled with a high level of accuracy (up to about 95%), even with a small set of CRPs. We repeat this experiment when the number of CRPs fed into the algorithm is reduced to 100. Our observation confirms that in this case, the accuracy of the model delivered by the algorithm drops significantly to approximately 80%. Note that although further reduction may result in obtaining an even less accurate model, from a practical point of view, it is not desirable. This is founded on the basis that a reduction in the number of CRPs exchanged in each round results in a more frequent reconfiguration of the hardware platform.

Hence, we shift our focus to designing rolling strategies changed in each round. In this regard, we employ two different rolling strategies: first, in each round of rolling only one stage of the PUF is reconfigured. More specifically, in four rolling rounds, the PUF configuration is changed from unrolled to a PUF with different rolled stages, denoted as Unrolled→5→9→17→22. For example, in the second round of rolling the configuration of the stage (5) is rolled back to "unrolled" and the stage (9) is reconfigured. In the second strategy, in each round, we have a various number of rolled PUF stages, e.g., Unrolled→(5,9)→(9,17) →(17,22)→(5,9,17,22). This strategy is much appropriate from the angle of view of practical implementation. Furthermore, we adopt the "refined" CRP management strategy, where the CRPs used in previous rounds are filtered (see Fig. 5(a)). The number of CRPs used in a round is 100, and the adversary can take advantage of all the CRPs observed by her: after the ith round, the number of CRPs used by the adversary is $100 \cdot i$. Additionally and similarly, the number of CRPs in the test set is $5000 \cdot i$. Table 3 shows the results of the experiments, in which the effectiveness of the rolling strategies is examined. As can be seen, the accuracy of the model is reduced drastically to approximately 65%.

6.3 Learning Combined PUFs

The goal of conducting this set of experiments is to assess the hardness of combined PUFs featuring either the XOR or the Tribes combiners. In our experiment, we set k=4, 6, and 12, i.e., the XOR Arbiter PUFs under test have 4, 6 or 12 chains, respectively. To compare the results for these combiners and the Tribes functions, we set the input length of the Tribes functions, k, to 4, 6, and 12 accordingly. Moreover, as suggested in [22], b, the size of the terms in the Tribes functions is set to 2 that is close to $\log_2 k$. To obtain somewhat hard PUFs

Rolling Strategy	Accuracy [%]			
	1 round of rolling	2 rounds of rolling	3 rounds of rolling	4 rounds of rolling
1	65.26	64.16	65.02	64.05
2	65.35	64.30	65	64.09

Table 3: Results for learning Arbiter PUFs rolled according to different strategies. In all of the experiments, the noise level is 0.05. The rolling strategy 1 refers to Unrolled→5→9→ 17→22, whereas strategy 2 is related to Unrolled→(5,9)→(9,17) →(17,22)→ (5,9,17,22).

Combination Function	Accuracy [%]			
	1 round of rolling	2 rounds of rolling	3 rounds of rolling	4 rounds of rolling
XOR (4)	88.42	88.37	86.41	85.41
Tribes (4,2)	65.04	63.84	63.53	63.47
XOR (6)	83.68	82.86	80.02	78.7
Tribes (6,2)	66.45	64.81	64.66	64.48
XOR (12)	77.2	76.90	76.95	76.89
Tribes (12,3)	63.3	57.1	54.72	53.34

Table 4: The accuracy of the model delivered by the low degree algorithm. In this table, the results are grouped into three main segments, showing how the results for the XOR and the Tribes functions should be compared to one another.

to be combined, the strategies regarding the rolling, and the CRP feeding in different rounds are employed, as what we have used in our experiments described in Sect. 6.2. Moreover, the number of CRPs in training and test sets are the same as the previous experiments on single PUFs. Table 4 presents the results of conducting the experiments described above. The core message conveyed here is that for both of the XOR-combined and the Tribes-combined PUFs the hardness is improved compared to the results for the standalone PUFs, even after rolling them (see Table 2). Another interesting aspect of the results is related to how the hardness of the combined functions is improved after rolling the PUFs in each round. For not only the Tribes-combined, but also the XOR-combined PUFs, after each round of rolling the hardness is slightly improved. However, in the latter case, even after 4 rounds, the hardness is still not satisfactory, compared to the Tribes-combined PUFs.

Comparison to the theoretical bounds: As discussed before, the hardness of the combined functions is at most equal the expected bias of the combination function (see Sect. 4). For Tribes functions, the expected bias can be approximated as $1/2 + 1/k$, when δ is close to $1/2$ [22]. For the XOR function, the expected bias can be calculated directly as $1/2 + 1/2(1 - \delta)^k$, where δ here denotes the hardness of the function f_{PUF}. According to the results provided in Sect. 6.2, on average, we can set δ to approximately 0.25. Hence, for k=4 the expected bias, and accordingly, the hardness of the XOR function is 0.6282, whereas it is 0.5890 for k=6. For the same setting, the expected bias of our Tribes function is approximately 0.75 and 0.6667, respectively. Consequently, the hardness of this function is *at least* equal to these values.

Nevertheless, as can be seen in Table 4, for the XOR function, the combined function may not be as hard as what can be approximated theoretically. This is a key difference between our practical setting and the theoretical one. The hardness amplification bounds provided for the XOR and the Tribes functions are established in a so-called *hard-core* setting [22]. This means that the adversary *knows* whether the output of the combined function is corrupted by the attribute noise or not (recall that the attribute noise may not always flip the output). On the contrary, in PUF scenarios, the adversary cannot gain this knowledge. In other words, we go beyond the hard-core setting and take the worst case scenario into account. Consequently, there is no one-to-one comparison between the bounds suggested in [22] and our results. Therefore, when XORing the PUFs, the hardness of this function cannot be tight or close-to tight for the hardness bound suggested by Yao's lemma. In contrast to this, the hardness achieved for

the Tribes functions in our experiments is much closer to what can be approximated in theory. This is regarding an intrinsic property of the Tribes function: remaining close to balanced, even under the effects of the attribute noise [22]. Hence, the Tribe functions can handle the noisy cases in real-world PUFs. For instance, for the Tribes (12,3), the expected bias is 0.5178, whereas it is 0.7689 for XOR (12). In conclusion, although one cannot directly compare our experimental results with the asymptotic hardness bounds of the Tribes and XOR functions, the hardness of the Tribes-combined functions can be closer to those bounds in practice.

7 Discussion

Comparison to the previous bound defined on the hardness of XOR Arbiter PUFs: We underline that the results for XOR Arbiter PUFs with a large number of chains ($k > \ln n$) do not contradict the upper bound established for provable algorithms, cf. [11]. The bound established in [11] holds for PAC learning algorithm in a general setting (i.e., no restriction on the distribution, regarding which we draw the CRPs); however, in this paper we discuss uniform PAC learning algorithms. Besides, the PAC learning algorithm considered in [11] has to deliver LTFs, whereas this is relaxed for the low degree algorithm. Thus, the results presented here are not comparable to what has been reported in the literature so far, e.g., [11].

Comparison to PUF protocols: Although our scheme shares some commonalities with protocols suggested in [23, 31], we emphasize the differences between them. First, for these protocols, the number of PUFs combined together using the XOR function has to exceed a lower bound defined for specific ML attacks. This is not sufficient since if the lower and/or upper bounds are not established in a general sense, an algorithm that has been overlooked up till now could break the security of a PUF built upon that lower/upper bounds cf. [7]. In contrast these protocols, our scheme is anchored on a solid foundation: the hardness amplification for Boolean functions, and bounds that hold in general, i.e., for any provable algorithm.

Another common aspect of our scheme and the PUF-based protocols is applying the notion of "controlled PUFs". For instance, the Slender PUF protocol [23] deploys an input network to not only restrict the attacker's access to the CRPs, but also meet the strict avalanche criterion (SAC) requirement. In our scheme, however, it is solely necessary to *filter* the CRPs used in the previous rounds of rolling. Moreover, as discussed before, the presence of the attribute noise in conjunction with the Tribes function ensures the maximum possible randomness, i.e., similar to the SAC property.

Comparison with schemes exploiting the impact of the noise: We stress that our architecture, enhanced by adding the attribute noise, should be clearly differentiated from schemes, which improve the security of a PUF by leveraging the impact of the *classification* noise (i.e., changing the output of a function by a random process), e.g., [32]. The latter approach is interesting, but it has been shown to be less effective than promised [29]. The core difference between such approaches and ours is that the attribute noise is harder to handle by provable machine learning algorithms, in particular, in the PAC learning framework [15]. Therefore, a stronger security guarantee can be given by our approach.

Resilience to attacks exploiting the impact of the noise: Although our scheme aims at making the provable ML attacks fail, it is worth considering noise-assisted ML attacks. Attacks, which take advantages the classification noise as a side channel information, mainly rely on the meta-stability condition at the measuring element of PUFs (e.g., the arbiter). In our scheme, the attacker has to deal with the attribute noise primarily, which could impair the effectiveness of the attacks suggested in [4, 8]. More precisely, the equations that have to be solved to build

a model of a PUF (cf. Equation 4b in [4] and the system of linear equations in Sect. VI.A and Sect. VI.B in [8]) cannot be formulated as required, due to the presence of the attribute noise.

Existence of the attribute noise in real-world implementations and how Tribes functions can deal with that: Upon implementing some instances of the same PUF on the same platform, ideally, the inter-distance between any pair of these instances should be close to 50%. However, this is not the case in practice due to the effect of the routing, and/or having not sufficient deviation in the manufacturing process variations from one instance to another. This can be precisely formulated by the impact of the attribute noise, which is not limited to our scheme featuring rolling. If the level of the attribute noise is sufficiently high to influence the bias of the PUF chains, clearly the desired hardness cannot be achieved, even for a large number of chains XORed together. Accordingly, as proved in theory and verified in practice, the final, combined PUF may not meet the close-to-balanced condition. Hence, the Tribes functions can be an appropriate combination functions to be used in not only our rolling-based scheme, but also in other schemes comprising a combination function for PUFs.

Aging can be considered and modeled as an example of the attribute noise as well. Each individual aged PUF circuit in a combined PUF, such as XOR PUF, might bias the overall responses of the combined PUF. In contrast to the XOR combination function, the Tribes functions can still deliver less biased responses over time. Moreover, under extreme aging conditions, new enrollments of CRPs in conjunction with using Tribes functions can recycle the PUF in an unbiased manner.

Enrollment phase: Since the influential stages can be different on different devices, the framework presented in Sect. 6 has to be applied to each new PUF during the enrollment phase. Thus, in addition to CRPs, the configuration of each rolled PUF have to be stored in a database for the future authentications.

8 Conclusion and Remarks

This paper aims at narrowing the gap between approaches for hardness amplification introduced in the ML theory and the PUF-related literature. Following the discussion on methods suggested to impair the effectiveness of ML attacks against PUFs, we conclude that two major classes of techniques can be distinguished. On the one hand, based on practical observations, to build a somewhat hard PUFs, rolling some stages of the PUF can result in failure for ML attacks. On the other hand, it is suggested to obfuscate the responses of PUFs by combining them together. Our proposed scheme takes advantage of both of these techniques, i.e., reconfiguration of some PUF stages and combining instances of a PUF implemented on the same platform. For this purpose, we make a marked paradigm shift, from the XOR function to the Tribes function, known for their random behavior. According to the proofs, the hardness of our proposed scheme implies that the attacker running provable algorithms cannot do better than filliping a coin to find the response of the PUF to an unseen challenge. This theoretical finding is further supported by conducting experiments on real-world PUFs in this paper.

9 Acknowledgments

The authors would like to thank the organizers of the Dagstuhl Seminar on Hardware Security (Dagstuhl-Seminar 16202), and especially, Prof. Dr.-Ing. Georg Sigl. In particular, the discussion session on "PUFs and Security Components" inspired us to explore the question of which other combining functions, rather than the XOR function, can be helpful in the

context of PUFs. We also thank Dr. Rainer Plaga from Bundesamt für Sicherheit in der Informationstechnik (BSI) for the fruitful discussion on the security of Arbiter PUFs that we have had in 2016.

Last but not least, the author would like to acknowledge the support of the National Science Foundation, CISE Community Research Infrastructure (CRI) Program under grant agreement No.1513239, National Institute of Standards and Technology under grant No. 60NANB16D248 and AFOSR under award number FA9550-14-1-0351.

References

[1] Ganji, Fatemeh and Tajik, Shahin and Stauss, Pascal and Seifert, Jean-Pierre and Forte, Domenic and Tehranipoor, Mark. Theoretical and Practical Approaches for Hardness Amplification of PUFs, 2019.

[2] D. Angluin and P. Laird. Learning from Noisy Examples. *Machine Learning*, 2(4):343–370, 1988.

[3] F. Armknecht, R. Maes, A. Sadeghi, O.-X. Standaert, and C. Wachsmann. A Formalization of the Security Features of Physical Functions. In *Security and Privacy, IEEE Symp. on*, pages 397–412, 2011.

[4] G. T. Becker. The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs. In *Cryptographic Hardware and Embedded Sys.*, pages 535–555. Springer, 2015.

[5] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis Dimension. *J. of the ACM*, 36(4):929–965, 1989.

[6] N. H. Bshouty, J. C. Jackson, and C. Tamon. Uniform-Distribution Attribute Noise Learnability. *Information and Computation*, 187(2):277–290, 2003.

[7] J. Delvaux. *Security Analysis of PUF-based Key Generation and Entity Authentication*. PhD thesis, Ph. D. dissertation, Shanghai Jiao Tong University, China, 2017.

[8] J. Delvaux and I. Verbauwhede. Fault Injection Modeling Attacks on 65 nm Arbiter and RO Sum PUFs via Environmental Changes. *Circuits and Sys. I, IEEE Trans. on*, 61(6):1701–1713, 2014.

[9] E. Fischer, G. Kindler, D. Ron, S. Safra, and A. Samorodnitsky. Testing Juntas. In *Foundations of Comp. Science, The 43rd Annual IEEE Symp. on*, pages 103–112, 2002.

[10] F. Ganji, S. Tajik, F. Fäßler, and J.-P. Seifert. Having No Mathematical Model May Not Secure PUFs. *J. of Cryptographic Engineering*, 2017.

[11] F. Ganji, S. Tajik, and J.-P. Seifert. Why Attackers Win: On the Learnability of XOR Arbiter PUFs. In *Trust and Trustworthy Comp.*, pages 22–39. Springer, 2015.

[12] F. Ganji, S. Tajik, and J.-P. Seifert. A Fourier Analysis Based Attack against Physically Unclonable Functions. In *Intl. Conf. on Financial Crypto. and Data Security*. Springer, 2018.

[13] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas. Controlled Physical Random Functions. In *Comp. Security Applications Conf.*, pages 149–160, 2002.

[14] S. Gehrer and G. Sigl. Using the Reconfigurability of Modern FPGAs for Highly Efficient PUF-based Key Generation. In *Reconfigurable Communication-centric Sys.-on-Chip, 2015 10th Intl. Symp. on*, pages 1–6. IEEE, 2015.

[15] S. A. Goldman and R. H. Sloan. Can PAC Learning Algorithms Tolerate Random Attribute Noise? *Algorithmica*, 14(1):70–84, 1995.

[16] T. M. Inc. MATLAB–The Language of Technical Computing. http://www.mathworks.com/products/matlab//.

[17] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas. A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications. In *VLSI Circuits, Digest of Technical Papers. Symp. on*, pages 176–179, 2004.

[18] R. Maes. An Accurate Probabilistic Reliability Model for Silicon PUFs. In *Cryptographic Hardware and Embedded Sys.*, pages 73–89. Springer, 2013.

[19] M. Majzoobi, F. Koushanfar, and S. Devadas. FPGA PUF Using Programmable Delay lines. In *Information Forensics and Security, IEEE Intl. WKSH on*, pages 1–6, 2010.

[20] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Techniques for Design and Implementation of Secure Reconfigurable PUFs. *ACM Trans. on Reconfigurable Tech. and Sys.*, 2, 2009.

[21] K. Matulef, R. O'Donnell, R. Rubinfeld, and R. A. Servedio. Testing Halfspaces. *SIAM J. on Comp.*, 39(5):2004–2047, 2010.

[22] R. O'Donnell. Hardness Amplification within NP. *J. of Computer and Sys. Sciences*, 69(1):68–94, 2004.

[23] M. Rostami, M. Majzoobi, F. Koushanfar, D. Wallach, and S. Devadas. Robust and Reverse-Engineering Resilient PUF Authentication and Key-Exchange by Substring Matching. *Emerging Topics in Comp., IEEE Trans. on*, 2(1):37–49, 2014.

[24] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling Attacks on Physical Unclonable Functions. In *Proc. of the 17th ACM Conf. on Comp. and Communications Security*, pages 237–249, 2010.

[25] D. P. Sahoo, D. Mukhopadhyay, and R. S. Chakraborty. Formal Design of Composite Physically Unclonable Function. In *WKSH on Security Proofs for Embedded Systems*, 2013.

[26] D. P. Sahoo, D. Mukhopadhyay, R. S. Chakraborty, and P. H. Nguyen. A Multiplexer-Based Arbiter PUF Composition with Enhanced Reliability and Security. *IEEE Trans. on Computers*, 67(3):403–417, 2018.

[27] D. P. Sahoo, S. Saha, D. Mukhopadhyay, R. S. Chakraborty, and H. Kapoor. Composite PUF: A New Design Paradigm for Physically Unclonable Functions on FPGA. In *Intrl Symp. on Hardware-Oriented Security and Trust*, pages 50–55. IEEE, 2014.

[28] A. Spenke, R. Breithaupt, and R. Plaga. An Arbiter PUF Secured by Remote Random Reconfigurations of an FPGA. In *Intl. Conf. on Trust and Trustworthy Comp.*, pages 140–158. Springer, 2016.

[29] J. Tobisch and G. T. Becker. On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation. In *Intl. WKSH on Radio Frequency Identification: Security and Privacy Issues*, pages 17–31. Springer, 2015.

[30] J. Ye, Y. Gong, Y. Hu, and X. Li. Polymorphic PUF: Exploiting Reconfigurability of CPU+ FPGA SoC to Resist Modeling Attack. In *Asian Hardware Oriented Security and Trust Symp.*, pages 43–48. IEEE, 2017.

[31] M. D. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, and I. Verbauwhede. A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication. *IEEE Trans. on Multi-Scale Comp. Sys.*, PP(99), 2016.

[32] M.-D. M. Yu, I. Verbauwhede, S. Devadas, and D. MRaihi. A Noise Bifurcation Architecture for Linear Additive Physical Functions. *Hardware-Oriented Security and Trust, IEEE Intl. Symp. on*, pages 124–129, 2014.