# Theoretical and Practical Approaches for Hardness Amplification of PUFs

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 robust against attacks, in particular, machine learning (ML) attacks. In 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". The goal of studies on the hardness amplification is to build a *strongly* secure construction out of considerably weaker primitives. This paper aims at narrowing the gap between these studies and hardware security, specifically for applications in the domain of PUFs. To this end, we first review an example of practical efforts made to construct more secure PUFs, namely the concept of rolling PUFs. Based on what can be learned from this and central insights provided by the ML and complexity theory, we propose a new PUF-based scheme built around the idea of using a new function, namely, the Tribes function, which combines the outputs of a set of PUFs to generate the final response. Our theoretical findings are discussed in an exhaustive manner and supported by the results of experiments, conducted extensively on real-world PUFs.

**Keywords:** Hardness Amplification · Complexity Theory · FPGA Security · Physically Unclonable Function · Partial Reconfiguration

## 1 Introduction

After the introduction of the first strong physically unclonable function (PUF) (i.e., a PUF with an exponential number of challenge-response pairs), it became soon evident that it is vulnerable to machine learning (ML) attacks [LLG+04]. 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. Among all potential solutions, a class of countermeasures relies on the *rolling* of 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., [SBP16, YGHL17].

Although applied on a trial-and-error basis and in a *blind* fashion, this approach can be helpful to improve the robustness of PUFs against ML attacks.

This can be explained by a fact, which has not always been paid attention to, that is 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 was first precisely noted as a *hidden variable* [Mae13]. It has been demonstrated that the hidden variables, namely a manufacturing process variable and a noise variable, account for the observable behaviors of a PUF cell, in particular, they reflect the underlying physical, random processes in a PUF stage.These variables have been reintroduced and reformulated in [GTS18] so that the noise model agrees with the respective model in ML theory. More specifically, it has been shown that the noise corresponding to the noise variable in a PUF stage is what has been called the "attribute" noise in the ML related literature. It has been further proven that even in the presence of this type of noise, there exists an ML algorithm, i.e., the low degree algorithm, to learn real-world PUFs [GTS18]. Consequently, one can deduce that the security of rolling PUFs can be broken by mounting the low degree algorithm. It is worth noting here that the low degree algorithm is one of the most efficient and effective algorithms in the sense that it is applicable even under noisy conditions. Moreover, since it is categorized as an improper learning algorithm, i.e., without limit on the hypothesis class, infeasibility of applying the low degree algorithm provides a strong security assurance.

Nevertheless, in order to employ the low degree algorithm, it is essential to have a bounded level of the attribute noise. Additionally, to obtain a sufficiently accurate model of a PUF with a high probability, a larger number of CRPs is required by ML algorithms, and similarly, by the low degree algorithm. Therefore, if a framework can be established to increase the level of attribute noise beyond an upper bound, an attacker applying the low degree algorithm can be stopped. In this regard, identifying some stages of a PUF, which are the best candidates for rolling to achieve this desired level of the attribute noise is a tedious task in practice, if possible at all. This is due to the challenges that one has to face to control the effect of the attribute noise, inevitably involving a process of trial and error. To help with this entire process, as another option, it is possible to limit the number of CRPs accessible to an adversary in each rolling round. We put emphasis on the fact that this method has been deployed in the PUF related literature and is usually referred to as "controlled PUF". However, among a wide range of possibilities to limit the access of the attacker to the CRPs, 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. Nonetheless, such an approach is still dependent on the practical aspects of the design, 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 aimed.

Seen from another perspective, the issue with ML attacks against PUFs has been addressed by adding components that account for non-linearity. As a prime example, the notion of XOR Arbiter PUFs has been introduced to tackle the problem with model-building attacks launched against Arbiter PUFs [SD07]. In this respect, the core idea behind using the XOR combination function was to obfuscate the output of an Arbiter PUF. Shortly afterward, it has been shown that using the XOR for combining multiple rows of parallel PUFs is beneficial to fulfill the strict avalanche criterion (SAC) [MKP08, MKP09]. Satisfying the SAC property assures that when flipping a single bit of a challenge, the response to the corresponding challenge changes drastically, i.e., each of the output bits flips with a probability of one half. Originated in Cryptography, in order to draw any conclusion on the robustness of a PUF against ML attacks, an analogy between the SAC property and a similar notion should be drawn. This has been addressed in [GTS18], where the notion of noise sensitivity has been reintroduced to assess the robustness of PUFs to ML attacks. More interestingly, a close connection between the noise sensitivity, the attribute noise, and the hidden variables, as hardware-related abstractions, has been established. Nonetheless, it is as yet unclear how and to what extent the *hardness* of a PUF, i.e., being mildly robust against ML attacks, can be amplified.

**Our Contribution.** The main contribution of this work is proposing a framework for the hardness amplification of PUFs[1]. We demonstrate that both practical and theoretical aspects of the problem are essential for achieving the desired, high level of hardness. In fact, these aspects are complementary: to meet the requirements imposed in our theoretical setting, it is inevitable to take into account the physical features offered by a real-world PUF. In working towards this goal, by taking effective practical measures, we add the attribute noise into our system to the extent that the PUF can be learned only with a low level of accuracy[2]. In doing so, we propose to roll a PUF, where 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. To this end, and to develop a systematic methodology, we employ the concept of influential stages, which has been introduced in *CHES'16* [GTFS16] to identify the influential PUF stages. In this case, we show that the reconfiguration of influential stages has the highest impact on obtaining fresh CRPs (i.e., their responses are flipped due to the attribute noise) from a rolled PUF. Afterward, we take into consideration the minimum number of CRPs, which an adversary requires to mount a provable ML algorithm to model a PUF for given levels of accuracy and confidence. Therefore, the designer can set the number of required rolling rounds upfront to prevent an attack.

Although being sufficiently secure for some less critical, lightweight systems, the above approach 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 functions in our scenario, where we enjoy the practical advantages offered by the rolling strategies, and therefore, the attribute noise [3]. Using the Tribes functions, famous for their randomness properties, is of twofold importance. First, these functions exhibit properties related to the noise sensitivity that make them a promising candidate to ensure a high level of hardness. Secondly, and even more interestingly, while the output of an XOR function can be dramatically biased in the presence of the attribute noise, the outputs of the Tribes functions remain balanced [O'D04]. Therefore, for schemes comprising a combination of PUFs in the presence of the attribute noise, using the Tribes functions is a central requirement to attain a desirably high level of security. We should here underline the fact that the presence of the attribute noise is not limited to a rolling-based PUF scheme, but when implementing a PUF, each and every stage of that can be affected by this type of noise [Mae13].

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 [MKD10]. 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.

[1]We stress that our framework in this paper addresses the passive ML attacks primarily, and other sophisticated side-channel and physical attacks are outside the scope of this paper.

[2]For a detailed discussion on how our scheme differs from that leveraging the *classification* noise, see Section 7

[3]In Section 7, we explain why the Tribes function should be used in schemes involving a combination function for PUFs.

## 2 Notation and Preliminaries

This paper covers several sub-fields in machine learning and complexity theory. Note that we stick to the notions used in each and every of these sub-fields and in this matter, caution and flexibility are needed.

### 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 $f_{\mathrm{PUF}} : \mathcal{C} \to \mathcal{Y}$, where $f_{\mathrm{PUF}}(c) = y$, describes a PUF. Ideally, for a given PUF $f_{\mathrm{PUF}}$, 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 $g_{\mathrm{PUF}}$ is *similar* to $f_{\mathrm{PUF}}$ [AMS+11]. We stress that our work does not cover the topics of formalization and formal definitions of the PUFs. For more details on these topics see, e.g., [AMS+11, AMSY16].

Similar to the most relevant studies on PUFs in the literature [GTFS16, GTS18], we represent PUFs as Boolean functions over the finite field $\mathbb{F}_2$. Let $V_n = \{c_1, c_2, \ldots, c_n\}$ denotes the set of Boolean attributes or variables, being either ***true*** or ***false*** denoted by "1" and "0", respectively. Furthermore, consider $C_n = \{0,1\}^n$ be the set of all binary strings with $n$ bits, and an *assignment* be a mapping from $V_n$ to $\{0,1\}$. Hence, an assignment can be thought of as an $n$-bits string, where the $i^{\text{th}}$ bit associated with the value of $c_i$ (i.e., "0" or "1").

A Boolean formula is a mapping that assigns values from the set $\{0,1\}$ to an assignment. Following this, each Boolean attribute is a formula, i.e., $c_i$ is a possible formula. Accordingly, a Boolean function $f : C_n \to \{0,1\}$ defines a Boolean formula, which can be represented by various classes of function. Nonetheless, 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 $f : \{-1,+1\}^n \to \{-1,+1\}$. 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 $f(c) = \operatorname{sgn}\left(\left(\sum_{i=1}^{n} \omega_i c_i\right) - \theta\right)$. 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** Among Boolean functions well-studied in the machine learning theory, Tribes functions introduced by Ben-Or et al. [BOL89] play an important role in Cryptography. 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, $b = O(\log k)$.

**Average Sensitivity of Boolean Functions** In order to define the Fourier expansion of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, again we use the encoding scheme $\chi(\cdot)$, defined above. Now the Fourier expansion of a Boolean function can be written as

$$f(c) = \sum_{S \subseteq [n]} \hat{f}(S)\chi_S(c),$$

where $[n] := \{1, \ldots, n\}$, $\chi_S(c) := \prod_{i \in S} c_i$, and $\hat{f}(S) := \mathbf{E}_{c \in \mathcal{U}}[f(c)\chi_S(c)]$. Here, $\mathbf{E}_{c \in \mathcal{U}}[\cdot]$ denotes the expectation over uniformly chosen random examples. The influence of variable $i$ on $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined as

$$\mathrm{Inf}_i(f) := \mathrm{Pr}_{c \in \mathcal{U}}[f(c) \neq f(c^{\oplus i})],$$

where $c^{\oplus i}$ is obtained by flipping the $i$-th bit of $c$. Note that $\mathrm{Inf}_i(f) = \sum_{S \ni i} (\hat{f}(S))^2$, cf. [O'D14]. Next, we define the **average sensitivity** of a Boolean function $f$ as

$$\mathrm{I}(f) := \sum_{i=1}^{n} \mathrm{Inf}_i(f).$$

**Noise Sensitivity of Boolean Functions** The term noise should not be mistaken as the notion of noise discussed in the PUF-related literature. The noise sensitivity of the Boolean function $f : \{-1, +1\}^n \to \{-1, +1\}$ can be defined as follows. Let $c$ be a string chosen randomly and uniformly. By flipping each bit of this string independently with probability $\varepsilon$ ($0 \leq \varepsilon \leq 1$) we obtain the string $c'$. The **noise sensitivity** of $f$ at $\varepsilon$ is

$$\mathrm{NS}_\varepsilon(f) := \mathrm{Pr}[f(c) \neq f(c')].$$

The noise sensitivity is closely related to the notion of noise stability, defined as follows.

$$\mathrm{Stab}_\rho(f) := 1 - 2\mathrm{NS}_\varepsilon,$$

where $\rho = 1 - 2\varepsilon$. One should notice the difference between the parameters based upon which the noise sensitivity and the noise stability are defined, namely, $\varepsilon$ ($\varepsilon \in [0,1]$) and $-1 \leq \rho \leq 1$.

## 2.2 Probably Approximately Correct Learning Model

A PAC learning algorithm [KV94b], similar to other ML algorithms, generates an approximately correct hypothesis, when it is given access to a set of *examples*. The main difference between a PAC learning algorithm and a conventional learning algorithm is that for the PAC learner, it is possible to ensure obtaining a desired hypothesis with a high probability. Defining this formally, consider the set $F = \cup_{n \geq 1} F_n$ denoting a target concept class, i.e., a set of Boolean functions over the instance space $C_n = \{0, 1\}^n$. Similarly, we can define a set of hypotheses $H = \cup_{n \geq 1} H_n$, which can be returned by the learning algorithm. 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 $H \subseteq F$, 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, $h \notin F$, cf. [DLSS13]. The first case is known as *proper* learning, whereas the latter is referred to as *improper* learning.

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 $C_n$. The hypothesis $h \in H_n$ that is a Boolean function over $C_n$ is an $\varepsilon$-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. In order to determine how many labeled examples $(c, f(c))$ are required by the PAC learner to provide the hypothesis, defined above, we should first define a measure related to the complexity of the target concept class. For an infnite concept class, i.e., with infnite cardinality, the Vapnik-Chervonenkis dimension $\mathrm{VC}_{\mathrm{dim}}(F)$ offers this measure [VC71]. Now we can define a uniform PAC learning algorithm as follows.

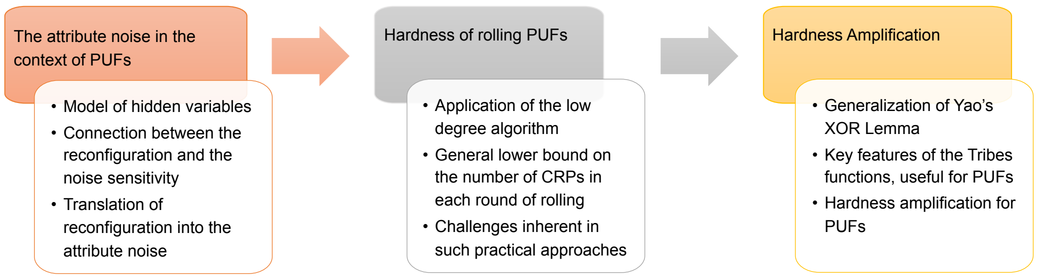
The attribute noise in the context of PUFs
Model of hidden variables
Connection between the reconfiguration and the noise sensitivity
Translation of reconfiguration into the attribute noise
Hardness of rolling PUFs
Application of the low degree algorithm
General lower bound on the number of CRPs in each round of rolling
Challenges inherent in such practical approaches
Hardness Amplification
Generalization of Yao's XOR Lemma
Key features of the Tribes functions, useful for PUFs
Hardness amplification for PUFs

Figure 1: The steps taken to propose a systematic hardness amplification methodology. Our approach benefits from a unique combination of practical and theoretical know-how.

**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 $\varepsilon$-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 general, for a PAC learning algorithm it is possible to establish a lower bound on the number of examples needed to obtain the $\varepsilon$-approximator for $f \in F$ with the probability at least $1-\delta$ [BEHW89]:

$$N = \Omega\left(\frac{1}{\varepsilon}\log\frac{1}{\delta} + VC_{dim}(F)\right).$$

## 3 From Fourier Analysis Based Attacks to Practical Hardness Amplification

The goal of the next two sections (Section 3 and Section 4) is to explain how the theoretical, well-known methodologies can be successfully applied in our case, and even further benefit from physical properties of a PUF. Figure 1 gives an overview of steps taken to achieve this goal. We begin with an observation made in [GTS18] that for the concept classes of known families of PUFs, even affected by the noise, there exists an ML attack relying on the principles of Fourier analysis, so-called the low degree algorithm. To launch this attack under the noisy conditions, the impact of the meta-stability in the PUF stages is considered as a case of the attribute noise. Nevertheless, this holds if the noise level does not exceed a certain upper bound [BJT03, O'D03]; otherwise, for desired accuracy and confidence level, the number of examples – the CRPs in our case – required to run the algorithm increases exponentially. What can be concluded from the above discussion is that in an attempt to stop an adversary running the low degree algorithm against a PUF, an increase in the level of the attribute can be a promising direction, as discussed in Section 3.2. Finally, we explain why this cannot be sufficiently effective in practice and propose a theoretically and practically superior method (see Section 4).

### 3.1 Translation of Reconfiguration into the Attribute Noise

In this section, we put our primary focus on how adding attribute noise can be thought of as a promising candidate to impair the effectiveness of provable algorithms applied to PUFs. From the point of view of PUFs, regardless of the origin of the noise, its impact can be on the stages of a PUF (e.g., a switching element in an Arbiter PUF) and/or the measuring element of the PUF (e.g., the Arbiter in an Arbiter PUF), if it exists [GTS18]. From the perspective of machine learning, the noise can be categorized as classification or attribute noise. The classification noise can be observed, when the output of a function is changed by a random process. On the other hand, the attribute noise deals with random changes in the inputs of the function. Moreover, compared to the classification noise, the attribute noise is harder to handle by provable machine learning algorithms, in particular, in the PAC learning framework [GS95]. Here we do not discuss the details of this categorization and refer the reader to [BJT03] on this matter; yet the relationship between attribute noise and the noise affecting a stage of a PUF is of great importance to our approach.

As demonstrated in [GTS18], the noise with an impact on a stage of a PUF accounts for the attribute noise. More specifically, the approach presented to show this is inspired by the model of hidden variables introduced in [Mae13]. This model considers the underlying physical processes in a (silicon) PUF stage, e.g., $i^{\text{th}}$ stage, namely the process variations $X_i$ $(1 \leq i \leq n)$ and the noise varying during each evaluation $N_i \sim \mathcal{N}(\mu_N, \sigma_N^2)$. It is known that $X_i$ follows a Gaussian distribution with the mean value $\mu$ reported by manufacturers as the nominal value, and the standard deviation $\sigma_i$, which is the result of the process variations, see for instance [MKP09]. For instance, these realizations correspond to the difference between the delays of crossed and straight signal paths in an Arbiter PUF. 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. Now consider two different evaluations of the PUF (with $c_i = 1$ and $c_i = 0$, respectively) that result in $z_{i,1} = x_{i,1} + n_{i,1}$ and $z_{i,0} = x_{i,0} + n_{i,0}$, depending on the challenge bit applied to the PUF. As can be understood, the realizations of the random variable corresponding to the noise are indicated by different indices to show that the noise realizations vary for two different evaluations of the PUF. Moreover, the realizations $z_{i,0}$ and $z_{i,1}$ contribute to the final response of the PUF [Mae13]. Under a meta-stable condition, the realizations $z_{i,0}$ and $x_{i,1}$ (and similarly, $z_{i,1}$ and $x_{i,0}$) can be very close to each other. In this case, it is not possible to distinguish if the challenge bit applied to the PUF is "0" or "1". Hence, the above condition is a case for the attribute noise.

This gives an answer to the question of how the attribute noise can be realized in practice [GTS18]. As a further step towards answering this, here we show that the reconfiguration of a PUF stage can be translated into the attribute noise. Although in this paper we focus on Arbiter PUFs, the following discussion is not limited to this family of PUFs. 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 [Mae13, GTS18], the realizations of the random variable $Z_i$ can be re-written to obtain $z_{i,0} = x'_{i,0} + n'_{i,0}$, where $n'_{i,0} = x_{i,0} - x'_{i,0} + n_{i,0}$. Similarly, we can define $y_{i,1}$, and consequently, $z_{i,1}$. 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. [GTS18]. It is straightforward to see that the conditions $x'_{i,0} = x_{i,1} - n'_{i,0}$, and $x'_{i,1} = x_{i,0} - n'_{i,0}$ meet the requirements for defining the attribute noise. In other words, in the above circumstance, it is possible to decide neither which configuration is used nor which challenge bit ("0" or "1") is applied to the PUF.

Following the procedure defined in [GTS18], the above uncertainty in the state of each PUF stage can be presented by a random variable with a Bernoulli distribution $A \sim D = \text{Bern}(\varepsilon)$, where $\varepsilon$ denotes the probability of being in an uncertain condition, as defined above (see Figure 2). Although the model developed above considers a single stage of a PUF, due to the definition of the attribute noise, it is easy to extend that to multiple stages of the respective PUF. To be more precise, the attribute noise affects each PUF stage independently, but by following an identical distribution $D$. Therefore, a random

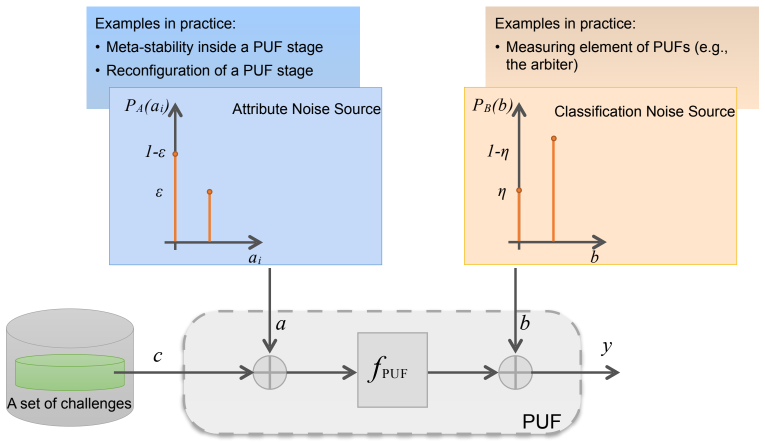
Examples in practice:
• Meta-stability inside a PUF stage
• Reconfiguration of a PUF stage
Attribute Noise Source
$P_A(a_i)$
1-ε
ε
$a_i$
Examples in practice:
• Measuring element of PUFs (e.g., the arbiter)
Classification Noise Source
$P_B(b)$
1-η
η
b
A set of challenges
c
a
$f_{\text{PUF}}$
b
y
PUF

Figure 2: The impact of the attribute noise and classification noise on the responses of a PUF. The impact of the attribute noise is modeled by a random string $a$ with independent random bits drawn according to a Bernoulli distribution. The random bit $b$ accounts for the classification noise and follows another Bernoulli distribution. This model is crucial to translate the impact of noise into the attribute and classification noises, known in ML.

string composed of independent random bits $(a_i)$ drawn according to the distribution $D$ reflects the impact of the attribute noise on all PUF stages. Now define the random string $a$ that denoted this string. Moreover, let the random bit $b$ drawn from the distribution $R = \text{Bern}\,(1 - \eta)$ represent the classification noise, cf. [GTS18]. It is straightforward to observe that the final response of the PUF can be formalized as $y = f_{\text{PUF}}(c \oplus a) \oplus b$. Note that we do not discuss the classification noise here while in our scheme, the issue with that is resolved by applying the majority voting technique. It is crucial to see the connection between this formalization and the definition of the noise sensitivity (see Section 2.1).

The next step is to translate the impact of the reconfiguration of a stage, related to the notion of the average sensitivity, to the attribute noise. To this end, Lemma 1 is of the essence in the sense that it states the connection between the random variables representing the attribute noise, the classification noise, and a challenge applied to a PUF, denoted by $A$, $B$, and $C$, respectively.

**Lemma 1.** *Let $U$, $D$ and $R$ denote the uniform, and two Bernoulli distributions*[4] *over the space $\{0, 1\}$. Additionally, the function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is an arbitrary Boolean function. The challenge $C \in_U \mathbb{F}_2^n$ is chosen uniformly. Besides, the attribute noise $A \in_D \mathbb{F}_2^n$ and the classification noise $B \in_R \mathbb{F}_2$ are represented by independent random strings and a random variable, respectively. It is straightforward to show that the random variables $(C, f(C \oplus A) \oplus B)$ and $(C \oplus A, f(C) \oplus B)$ follow identical distribution.*

This simple, but useful, lemma implies that the impact of the noise on the stages of a PUF can be taken into account by the identically distributed variable $(C \oplus A, f(C))$, i.e., the well-known model of the attribute noise, cf. [BJT03]. In a nutshell, here we make a connection between the learnability of the attribute noise, the average sensitivity, and the effect of reconfiguring a PUF stage.

[4]For PUFs, it is known that $D$ and $R$ are Bernoulli distributions [Mae13, GTS18], although the lemma holds in general case, where these distribution can be arbitrary.

## 3.2 How Hard Is It to Learn Rolling PUFs?

After elaborating on how the reconfiguration can be seen as a case for the attribute noise, we should answer the following questions. First, how hard is it to learn the rolling PUF, i.e., a PUF that is repeatedly reconfigured? Answering this question is straightforward since we have already established a relationship between the notion of reconfiguration and the attribute noise in PUFs. Moreover, which algorithm should be considered in this matter? Among different ML algorithms, we focus on the low degree algorithm. The crucial aspects of this ML algorithm, so-called low degree algorithm or LMN-style algorithm [LMN93], are two-fold. First and foremost, the low degree algorithm can be categorized as an *improper* PAC learner (see Section 2.2). Under this learning scenario, the freedom given to the algorithm to deliver a hypothesis $h \notin F$ can lead to a more powerful learning algorithm, compared to proper learning algorithms [DLSS13]. Clearly, one can deduce that if the improper learning is intractable for a target concept class, most likely no proper learning algorithm for that class can be devised either. The second interesting aspect of the low degree algorithm is related to how this algorithm deals with noisy examples. It has been demonstrated that this algorithm is applicable even under scenarios, where not only *classification*, but also *attribute* noises are present [BJT03].

The rationale behind the low degree algorithm is that various classes of Boolean functions can be approximated by taking into account solely a small number of their Fourier coefficients (so-called "low" coefficients), corresponding to small subsets of $[n]$ (see Sec. 2.1).

**Theorem 1.** (**Low degree algorithm**) [Man94,LMN93,O'D03] *Assume that an algorithm can determine a set* $\mathcal{S} \subseteq 2^{[n]}$ *containing subsets of* $[n]$ *so that* $\sum_{S\in\mathcal{S}} \hat{f}(S)^2 \geq 1-\varepsilon$. *The algorithm is given a pre-defined confidence level* $\delta$ *and access to a polynomial number of input-output pairs of the Boolean function* $f$ *that are chosen uniformly at random. With probability* $1-\delta$ *the algorithm delivers a Boolean function* $h$ *that is an* $\varepsilon$*-approximator of the Boolean function* $f$ *such that*

$$\sum_{S\subseteq[n]} \left(\hat{f}(S) - \hat{h}(S)\right)^2 \leq \varepsilon.$$

*The running time of the algorithm is poly* $(|\mathcal{S}|, n, 1/\varepsilon, \log_2(1/\delta))$.

For the proof of this theorem, we refer the reader to [Man94, LMN93], in which the mechanism for determining the set $\mathcal{S}$, and the lower bound on the number of input-output pairs required by the algorithm has been discussed extensively. In addition to Theorem 1, what has been proven by Bshouty et al. is of great importance to our approach. They have shown that concepts that are uniformly PAC learnable by adopting the low degree algorithm can be further learned under noisy conditions [BJT03]. This can be formulated precisely in the following theorem.

**Theorem 2.** [O'D03] *Let* $\alpha : [0,1/2] \to [0,1]$ *be a strictly increasing continuous function. For any given Boolean function* $f$ *in the target concept class* $F$ *with* $NS_\varepsilon(f) \leq \alpha(\varepsilon)$. *We have*

$$\sum_{|S|\geq m} \hat{f}(S)^2 \leq \varepsilon,$$

*where* $m = 1/\alpha^{-1}(\varepsilon/2.32)$ *and* $\alpha^{-1}(\cdot)$ *denotes the inverse of the function* $\alpha(\cdot)$. *There exists a uniform-distribution learning algorithm for* $F$ *that requires poly* $(n^m, \log_2(1/\delta))$ *number of examples to deliver a Boolean function* $h$ *that is an* $\varepsilon$*-approximator of* $f$ *with the probability at least* $1-\delta$.

Note that while we do not consider the classification noise here, the above mentioned bound reflects the impact of the attribute noise solely. The proof of Theorem 2 is

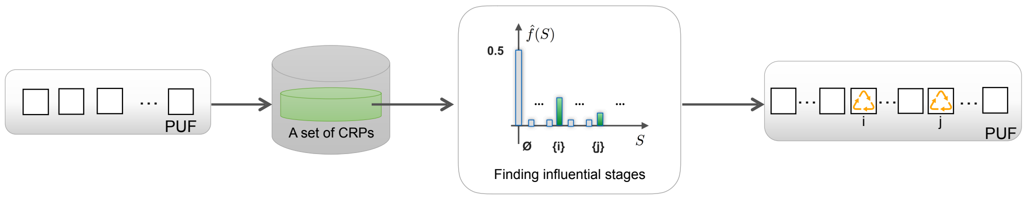

Figure 3: From a conventional PUF to its rolled version. First, the designer should collect a sub-set of CRPs to determine the influential stages of the PUF [GTFS17]. To obtain the rolling PUF, various strategies can be employed (see Section 6).

straightforward (see Corollary 2.3.3 and Theorem 5.2.1 in [O'D03] as well as Section 5 in [BJT03]), although the messages conveyed by that need special attention. As can be understood from Theorem 2, the existence of the attribute noise leads to an exponential increase in the number of examples required by the algorithm. In contrast to this, the impact of the classification noise on this number of examples is polynomial in the level of the classification noise. Hence, it is tempting to wonder whether by introducing the attribute noise to a PUF, the problem of learning can become intractable.

To this end, we suggest that some stages of the PUF can be *rolled* in several rounds to obtain a more ML attack-resistant primitive. The concept of rolling PUFs is inspired by the key rolling known in Cryptography, although the commonality between them is limited to the change in some security-related element of the system once using that. After translating the impact of reconfiguration into the attribute noise, now it is evident that a *blind* partial reconfiguration cannot automatically assure obtaining fresh CRPs that are affected by the attribute noise. 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 [Qui86, AL88]. Therefore, before rolling a PUF, it is crucial to determine the influential stages, for instance, by applying an approach proposed in [GTFS16, GTFS17], see Figure 3.

Furthermore, in our scenario, the reason behind changing the architecture (i.e., here, the configuration) and CRPs of the PUF in several rounds is two-fold: first, if the rolled stage is used in a legacy PUF (i.e., with no rolling round), it can be still possible to learn the PUF. To provide a better understanding of this, assume that a PUF designer implements a PUF and runs some tests to assess the security of the PUF against ML attacks. Afterward, to achieve better resistance against these attacks, the configuration of some stages are changed by the designer. This change can be translated to an attribute noise (see Section 3.1), however, if the noise sensitivity of the Boolean function representing the PUF ($\alpha(\varepsilon)$) is bounded by some small value, it is still possible to learn the rolled PUF.

Secondly, we put emphasis on the fact that in our design not only the architecture, but also the set of CRPs used by the scheme is changed in each round of rolling. 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 (see, e.g., [GCVDD02]). Our design goal is to stop an attacker attempting to collect CRPs, even being noisy, to come up with a sufficiently accurate model of the challenge-response behavior of the PUF, i.e., an $\varepsilon$-approximator of $f_{\text{PUF}}$ with a pre-defined, desired level of confidence ($\delta$). Therefore, in each round of rolling, the attacker should not have access to a set of CRPs needed to build such a model. According to the theory of machine learning, to address this issue, a lower bound on the number of examples required by the learning algorithm can be established (see Section 2.2).

To put it differently: the importance of defining this lower bound is attached to the fact that we should compute the number of CRPs allowed in each round. To establish this bound, we should consider the representation of the target concept class that is, for instance, an LTF for an Arbiter PUF. It is worth noting that our focus on improper PAC learning does not rule out the possibility of defining a representation of the *target concept* class. As for LTFs with $n$ Boolean variables the Vapnik-Chervonenkis dimension is $n+1$, it is easy to calculate the lower bound as $N = \Omega\left(1/\varepsilon \log(1/\delta) + n + 1\right)$. 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. However, a bound being more tight to the upper bound can be established for a learning algorithm used in our approach. For the low degree algorithm, the lower bound equals

$$\Omega\left(\frac{1}{2}\sqrt{\frac{n^d}{\varepsilon}}\ \ln\left(\frac{2n^d}{\delta}\right)\right),$$

where $d = 1/\varepsilon^2$, cf. [Man94, O'D03]. At first glance, it may seem that no reliance is placed on the representation of the target concept class, when the lower bound on the number of CRPs for the low-degree algorithm is established. Nevertheless, we stress that this bound holds for any Boolean function exhibiting $\mathrm{NS}_f \leq \sqrt{\varepsilon}$, which is true for LTFs with $\mathrm{NS}_{\mathrm{LTF}} = O(\sqrt{\varepsilon})$ [KOS02]. More interestingly, both upper bounds discussed above are achieved for the noiseless setting. In other words, in the presence of noise, the accuracy of the model delivered by the algorithm is less than $\varepsilon$ for the same number of CRPs and the confidence level. Therefore, if we stick to the lowest upper bound mentioned above, i.e., $N$, and introduce noise to our scheme, in none of the rounds the adversary can obtain a model that is much more accurate than flipping a coin.

## 4 Theoretical and Practical Hardness Amplification

Although from a theoretical point of view the design discussed in Section 3.2 seems solid, in practice the desired hardness may not be achieved easily. This is due to the level of the attribute noise, which can guarantee that the Fourier coefficients cannot be approximated for a given number of examples, with sufficiently high accuracy. As proven in [BJT03, O'D03], if $m$ (see Theorem 2) does not exceed the upper bound $O(\log n)$, it is still possible to approximate the coefficients, given a polynomial number of examples. Hence, it is crucial to shift our focus to other approaches depending less on the practical aspects of fabrication, which are not entirely under our control.

In machine learning theory as well as complexity theory, hardness amplification is concerned with addressing the following question. Is it possible to create a significantly hard problem from a moderately hard one? Equivalently, and naturally, this can be formulated in the sense of security. In other words, given a weakly secure construction of a cryptographic primitive, we are interested in building a strongly secure construction out of it. This problem has traditionally been discussed for one-way functions [Yao82], where, intuitively, a combination of independent repetitions of a weakly one-way function can be converted to a much harder, strong one-way function. This result has been widely acknowledged and adapted in other cases, for instance, for decision problems the hardness amplification is formulated as Yao's XOR lemma [DJMW12, GNW11]. More precisely, this lemma asserts that if a Boolean function $f$ is weakly unpredictable within some complexity bound, for sufficiently large $k$, depending on the complexity bound, the function $F(x_1, ..., x_k) := \bigoplus_{i=1}^{k} f(x_i)$ is virtually unpredictable within the respective complexity bound. This means that the mechanism of generating the responses of $F$ is not essentially better than flipping a coin, cf. [GGR98]. This lemma was first proven by

Levin [Lev87] and the proof has been carried out in a uniform model of complexity[5]. We can now provide a more precise definition of Yao's XOR lemma.

**Theorem 3.** (**Yao's XOR lemma**) [Tre03] *Consider $f : \{0,1\}^n \to \{0,1\}$ that is a Boolean function that is hard to compute on $(1-\delta)$ fraction of its input*[6]*. Then, computing $f(x_1) \oplus ... \oplus f(x_t)$ on more than $1/2 + \varepsilon$ fraction of its inputs (i.e., $x_1, ..., x_k$) is hard, where $\varepsilon$ is roughly $(1-\delta)^k$.*

One can observe a close relationship between the concept of XOR Arbiter PUF and Yao's lemma, although it has not even 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 imposed by machine learning attacks against PUFs. As suggested in [SD07], in order to prevent such attacks, XORing multiple outputs of a PUF can to some extent obfuscate the outputs of that PUF. One of the well-received primitives constructed according to this notion is XOR Arbiter PUFs. Nevertheless, it has been shown that this can be only marginally effective since empirical and provable attacks are launched to compromise their security [RSS$^+$10, GTS15]. More importantly, it has been proven that in order to prevent machine learning attacks compatible with provable frameworks, the number of Arbiter PUFs XORed together should exceed an upper bound, namely, $\ln(n)$. Despite these efforts, a precise formulation of Yao's XOR lemma in the PUF-related literature is missing. To address this, here we should take some steps towards providing the setting that is required to transform the PUF-related scenario to one being compatible with Yao's lemma. In this regard, the following aspects of this lemma should be carefully addressed.

**Hardness on Average:** We should underline the importance of the difference between the average-case and the worst-case hardness assumptions. Although worst-case hardness has attracted a great deal of attention in the complexity theory and has been seen as a commonly-used hardness metric, the average-case hardness plays a crucial role in not only complexity theory, but also cryptography. Intuitively, the worst-case hardness assumption states that a problem is hard on some fraction of the inputs, whereas the average-case hardness refers to a case that the problem is hard on *most* of its inputs. It is known that for a given problem, one can conclude the worst-case hardness from the average-case hardness, and therefore, the average-case hardness is a stronger assumption. More formally, given a problem, in contrast to assuming that there is no algorithm solving an $\varepsilon$ fraction of the inputs, we require that no algorithm can solve $1/2 + \varepsilon$, or more, of the inputs [Yao82].

In the context of PUFs, the most relevant approaches to notice are developed in the PAC learning framework [GTS18]. The close connection between this learning framework and complexity theory has been established by the seminal work of Kearns and Valiant [KV94a]. More specifically, they have demonstrated that the hardness of improper PAC learning (i.e., no restriction on the hypothesis class) can be proven based upon hardness on average assumptions cf. [DLSS13]. Interestingly, it has been further proven that for some representation class, the hardness of learning holds even in the sense of weak learning [KV94a]. According to the above discussion, it is very natural to take into account the average-case hardness here.

**The Complexity Class:** Virtually all existing lower bounds known for improper learning are derived based on the assumption made about the hardness of some cryptographic schemes, e.g., RSA cryptosystem [KV94a]. In contrast to such challenging approaches, it

[5]There is a uniform Turing Machine that runs the same algorithm regardless of the size of the input.

[6]This parameter should not be confused by the confidence level defined in the context of PAC learning framework. Here we stick to the standard notion used in complexity theory-related literature.

has been demonstrated that in the average-case sense, the hardness of improper learning problems can be proven in a more straightforward manner [DLSS13]. At the core of this is the assumptions about Constraint Satisfaction Problems (CSPs) that are made to show the (in)tractability of some PAC learning problems. In particular, Daniely et al. have proven that *improper* learning of polynomial-size Disjunctive Normal Forms (DNFs) is hard *on average* in the PAC learning framework [DLSS13]. According to the result presented in [DLSS13], a DNF with the size exceeding $n \log(n)$ is hard on average. This has been obtained by making a hardness assumption about the hardness of refuting a CSP, namely, random 3-SAT problem [DLSS13][7]. As a $k$-junta function can be formulated as a DNF of size $2^k$, the above result holds, of course, for learning $k$-juntas, where $k > \log(n) + \log(\log(n))$. Nonetheless, for a small, constant $k$, the problem of learning $k$-juntas is solvable in polynomial-time. Accordingly, the problem of learning PUFs represented by small $k$-juntas, e.g., Arbiter PUFs, is within NP. As can be understood from the discussions in Section 3.2 and Section 4 we interchangeably use LTF and $k$-junta functions to represent Arbiter PUFs. This is due to the fact that for sufficiently small $\varepsilon$, as desired in PUF scenarios, any LTF can be close to a $k$-junta [Bou02, DJS+15].

**Hardness Amplification within NP:** Being motivated by the fact that Yao's XOR Lemma can be useful for the hardness amplification for EXP (i.e., problems with exponential run time), O'Donnell has generalized this lemma so that it can be applied within the complexity class NP [O'D04]. In fact, Yao's lemma is not applicable in this complexity class since $F(x_1, ..., x_k) := \bigoplus_{i=1}^{k} f(x_i)$ may not be in NP, even if $f(\cdot)$ is in NP. To address this issue, the following theorem has been proven by O'Donnell.

**Theorem 4.** (**Hardness Amplification within NP**) [O'D04] *(Less formal, cf. [HVV06]) Let the function $f : \{0,1\}^n \to \{0,1\}$ denotes a Boolean function in NP that is $1/poly(n)$-hard to be computed. There exist a function $F(x_1, ..., x_k) := C(f(x_1), \cdots, f(x_k))$ in NP that is $1/2 + n^{-1/3+\varepsilon}$ hard, where $C$ is a combination function*[8].

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 [O'D04], 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 $C(x) \geq C(y)$. This property ensures that the combined function $F$ *remains* in NP. Moreover, the combination function $C$ should exhibit low *noise stability*, or equivalently, low expected bias. To introduce such combination function, O'Donnell has proposed functions, for which each Boolean variable has a small influence, as suggested by Ben-Or and Linial [BOL89]. More specifically, the "Tribes" function has been used to prove Theorem 4. The inherent feature of the Tribes functions making them suitable for the hardness amplification is not only their monotonicity, but also being close-to-balanced, and in particular, remaining close-to-balanced even in the presence of the attribute noise.

## 4.1 Hardness Amplification for PUFs

Now we can eventually shift our focus to hardness amplification for PUFs. We argue that the hardness amplification following the Theorem 4 is much stronger than what can be achieved by XORing the PUFs. First and foremost, as discussed in the previous section, the notion of hardness on average offers a much stronger assumption, compared to worst-case ones. Moreover, and more crucially, it could be doubted if the legacy Yao's XOR lemma (or in other words, using the XOR combination function) is sufficient since it is not required that the combined function remains in NP. Although we are not interested in amplifying the hardness within the NP necessarily, the new combination

[7]Note that in this setting the examples are drawn uniformly.

[8]We adopt the theorem that holds also for unbalanced functions.

functions, i.e., Tribes functions, provides an inherently distinctive characteristic that makes them attractive in PUF scenarios, namely being more noise stable. On the one hand, Tribes functions are sufficiently noise sensitive to ensure a close-to-random behavior of the combined function, whereas on the other hand in the presence of the attribute noise, they are more robust. In other words, as precisely formulated in [O'D04], the latter characteristic of the Tribes functions is related to being nearly balanced and preserving this quality even when being subject to the attribute noise.

Last but not least, we should provide the combination function (the XOR or the Tribes function) with $k$ outputs of the function $f$, namely $f(x_1), ..., f(x_k)$, as required by Theorem 3 and Theorem 4. In our PUF construction, similar to the known, widely used XOR Arbiter PUF, the inputs $x_1, ..., x_k$ are the same. To resolve the difference between our setting and the requirement imposed by the hardness amplification theorems, we apply Lemma 1. In conjunction with this lemma, one can observe that our scheme takes advantage of the attribute noise introduced to different instances of the PUF. The immediate conclusion that can be made from this is twofold: first, for each challenge chosen uniformly and fed into an Arbiter chain, e.g., $i^{\text{th}}$, the distribution $A_i$, is different due to the manufacturing process variations (see Section 3.1). Therefore, the inputs $C \oplus A_i$ $(1 \leq i \leq k)$ are different. Secondly, the impact of the attribute noise on the input of each chain $C \oplus A_i$ $(1 \leq i \leq k)$ is statically the same as its impact on the output of the chains $f(C \oplus A_i)$. Hence, in the presence of the attribute noise, having $k$ instances of the PUF given the same challenge, the responses of the PUFs mimic the characteristics required for $x_1, ..., x_k$. We are now in a position to take the next step by stating the following theorem.

**Theorem 5.** *Consider a PUF $f_{PUF} : \{0,1\}^n \rightarrow \{0,1\}$ that is given $N$ pairs of CRPs chosen uniformly and an adversary using these $N$ pairs to come up with an $\varepsilon_N$-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 $\delta$, and $N$ being the minimum number of CRPs that a polynomial-time algorithm required to deliver an $\varepsilon_N$-approximator of the function $f_{\text{PUF}}$ with the desired probability $1 - \delta$. Clearly, according to the PAC model, $\varepsilon_N$ is a function of $\delta$, $N$, and $n$. More precisely, the achievable accuracy level is indeed $1/poly(n)$ (see Section 2.2). Note that with this number of CRPs no polynomial-time algorithm can achieve better than an $\varepsilon_N$-approximation of $f_{\text{PUF}}$. The proof is completed by taking Theorem 4 and setting $k = O(n^{2/3})$.

■

We stress that the above theorem and its proof can be thought of as a simplified version of the theorem proved in [FLS11]. Here we set out the high-level principles of the approach presented by Feldman et al. They essentially have proven that O'Donnell's hardness amplification method [O'D04] can be adopted within the uniform-distribution PAC model to amplify the hardness of "mildly hard" learning problems. Furthermore, as can be understood from Theorem 5 and its proof, although in Section 3.2 the low degree algorithm is considered as an example of provable algorithms, in this section we do not impose any limitation on the provable algorithm applied by an attacker. More particularly, Theorem 5 is formulated based on the properties of neither the low degree algorithm nor any specific provable algorithm.

## 5 PUF Implementation

As discussed before, the concept of rolling a PUF can assist us to make more secure PUFs. One approach to roll a PUF circuit on an application-specific integrated circuit (ASIC) or FPGA is to have multiple PUF instances in parallel, where they are selected by a multiplexer [SMCN18]. Another approach is to physically swap a PUF circuit partially

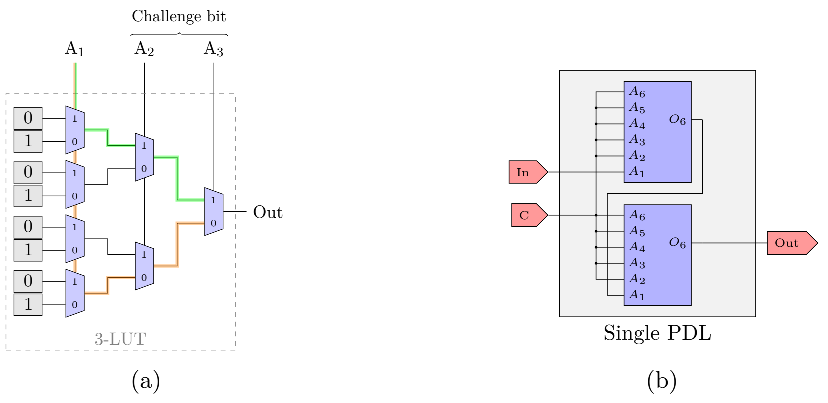

Figure 4: (a) Illustration of the propagation delay in a 3-input LUT based on the state of the challenge bit. (b) Composition of the PDL from two 6-input LUTs which operate as inverters. `C` denotes the challenge bit. `In` and `Out` are the input and output for the propagating signal, respectively.

or entirely, which is only feasible by employing the dynamic reconfigurability feature of mainstream FPGAs [SBP16]. To have the same flexibility on ASICs, eFPGAs [Ach18,Fle18] can be helpful, since they can be integrated into ASICs along other IP cores. Dynamic reconfigurability makes the reconfiguration of certain resources of FPGA during runtime possible, which can be beneficial for saving area and power in several applications. Critical security circuits, such as cryptographic algorithms and PUFs, can take advantage of dynamic reconfiguration capability as well. For instance, it has been reported that adding a random number of registers through reconfiguration, cryptographic algorithms, such as AES, can become side-channel and fault attack resistant [MGV08,SMMG15]. Later, similar ideas have been applied to PUFs to make them either ML resistant [SBP16, YGHL17] or increase their response entropy [GS15, BS17].

The first introduced rolling PUF [SBP16] has utilized the remote reconfiguration capability of a Field Programmable Gate Array (FPGA) to reconfigure the entire PUF circuit around the FPGA. However, inside an FPGA, where the resources are shared among several applications, we might not be able to move the PUF around the FPGA freely. Moreover, implementing a strong PUF, such as Arbiter PUF or Bistable Ring 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 setting, namely, only the influential stages can be reconfigured in each round of rolling.

In this section, we describe the setup used to realize the approach discussed in Sect. 3. we have realized our rolling PUF using the partial reconfiguration feature of a common FPGA.

## 5.1 FPGA Implementation of Arbiter PUF

To realize a configurable PUF, we deployed 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 it on an FPGA [MKD10]. Classical Arbiter PUF implementations utilize two digital multiplexers per stage of the PUF. As each multiplexer

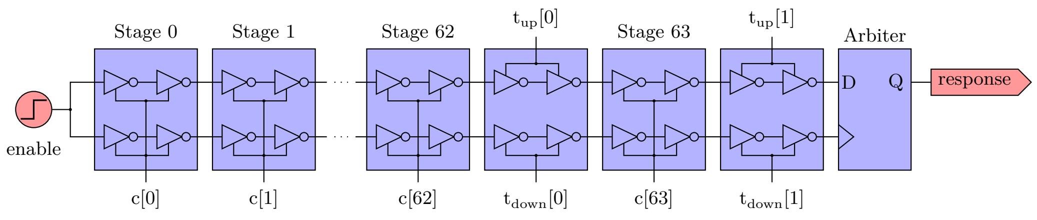

Figure 5: Schematic structure of a PUF using PDLs as stages.

is realized by a Lookup Table (LUT), two inputs out of four available inputs of LUT are used. Based on *don't-care* inputs, the output of multiplexer can be loaded from different SRAM cells inside the LUT and take different routes to the output. This fact leads to different propagation delays, and therefore, delay imbalances for the two PUF routes. Hence, due to routing constraints in a LUT of an FPGA, we implemented the stages by two independent chains using so-called programmable delay line (PDL) as in [MKD10], see Figure 5.

In contrast to implementations of a path switching structures with multiplexers, where routing constraints make it very difficult to adjust the nominal delays between the upper and lower paths, the use of PDLs allows an almost symmetrical structure. 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. An example of such configuration in a 3-input LUT is illustrated in figure 4a. The signal applied to $A_1$ takes different distances, depending on the assignment of the inputs of $A_2$ and $A_3$. If $A_2 = 1$ and $A_3 = 1$, the upper highlighted path is selected, see figure 4a; Otherwise, if $A_2 = 0$ and $A_3 = 0$, the lower and longer path is chosen. Similarly, 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. A hint that the physical paths of the LUT inputs within the Xilinx's 7-Series can cause different signal propagation times due to their structure can already be found in the Xilinx documentations [Xil], in which the inputs $A_6 - A_5$ are described as the fastest. In this case, the least delay with a challenge bit zero and the largest with one can be generated. We have programmed each LUT with the following Boolean expression:

$$O_{LUT} = \neg I_1 \vee (I_2 \wedge I_3 \wedge I_4 \wedge I_5 \wedge \neg I_6)$$

where, $I_1 - I_6$ are the connected signals to $A_1 - A_6$ and $O_{LUT}$ is the output signal of the LUT. The resulting truth table generates the desired behavior for the entries where $I_2 - I_6$ take the same values. By implementing a second inverter connected in series, the caused signal inversion is reversed, see figure 4b. In principle, instead of an inverter, the implementation of the identity function would also be conceivable, which would require only one LUT for a PDL. However, there is a risk that the LUT is completely optimized and trimmed by the netlist compilation due to the lack of logical added value.

Our design consists of a 64-bit Arbiter PUF, see figure 5. 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, for the upper and lower paths of the PUF. For constructing a PUF with a 64-bit challenge, 64 of these modules are chained in series. Additionally, there are a certain number of tuning stages [MKD10], 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 to get a symmetrical design. In principle, the tuning stages can be inserted at any place. At the end of the chain, a D-flip-flop serves as an arbiter, where data input is connected to the upper path and at the clock input to the lower path. The response of the PUF is zero if the total delay through all stages of

| LUT input | Equation | LUT config. |
|---|---|---|
| $A_1$ | $O_{LUT} = \neg I_1 \vee (I_2 \wedge I_3 \wedge I_4 \wedge I_5 \wedge \neg I_6)$ | 0x55555555D5555555 |
| $A_2$ | $O_{LUT} = \neg I_2 \vee (I_1 \wedge I_3 \wedge I_4 \wedge I_5 \wedge \neg I_6)$ | 0x33333333B3333334 |
| $A_3$ | $O_{LUT} = \neg I_3 \vee (I_1 \wedge I_2 \wedge I_4 \wedge I_5 \wedge \neg I_6)$ | 0x0F0F0F0F8F0F0F0F |
| $A_4$ | $O_{LUT} = \neg I_4 \vee (I_1 \wedge I_2 \wedge I_3 \wedge I_5 \wedge \neg I_6)$ | 0x00FF00FF80FF00FF |
| $A_5$ | $O_{LUT} = \neg I_5 \vee (I_1 \wedge I_2 \wedge I_3 \wedge I_4 \wedge \neg I_6)$ | 0x0000FFFF8000FFFF |

Table 1: Combinations of different physical LUT inputs.

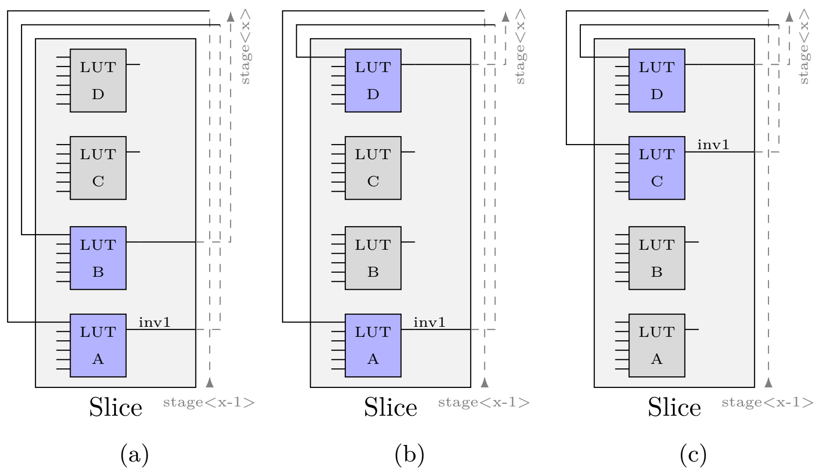

Figure 6: Examples of rolling the PUF stages with different LUT combinations in a slice: (a) using LUTs A and B for an unrolled stage, (b) swapping one the LUT B with the LUT D, (c) Swapping both LUTs A and B to LUTs C and D.

the upper signal path is greater than that of the lower; otherwise, the response is one.

## 5.2 Partial Reconfiguration of PUF

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 [GS15], 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 figure 6) 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 figure 7. The logical function of the LUT must be adapted in order to obtain an inverter behavior again, see Table 1. However, it does not make sense to use input $A_6$, as this would not result in two substantially different propagation paths.

## 5.3 Measurements

For each experiment, we collect CRPs from the reference PUF and the rolled one obtained by applying one of the mentioned approaches in section 5.2. The tuning stages are adjusted in a way that a ratio of nearly 50% of zeros and ones is obtained. Due to very little delay differences between two chains, the arbiter can sample a meta-stable signal. Therefore, for each given challenge, we conducted 19 response measurements and applied majority voting

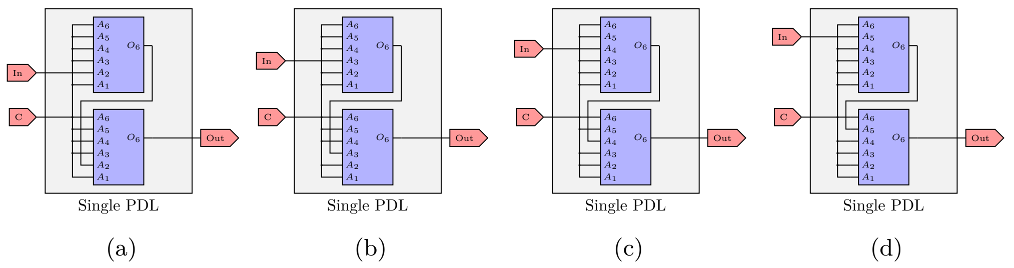

Figure 7: Rolling stages of the PUF by using different physical LUT inputs as propagation path. In a 6-input LUT, four new reconfigurations can be obtained, namely changing the input from $A_1$ to (a) $A_2$, (b) $A_3$, (c) $A_4$, and (d) $A_5$. `C` denotes the challenge bit and `In` the input port for the propagating signal.

on the gathered responses. After majority voting of each given challenge the number of '1's and '0's are counted, and if the number of majorities is less than 14, 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 $\varepsilon$ and $\delta$. 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 [Inc].

## 6 Results

In this section, we present our results, outlined as follows (see Figure 8). First and foremost, we provide the results related to applying a $k$-junta testing algorithm and discuss them in details. Secondly, we show how rolling a PUF can affect the average sensitivity of the Boolean function representing that PUF. We discuss to what extent Friedgut's theorem [Fri98] can help us to come up with an appropriate representation of the PUF. In conjunction with these results, we provide insight into whether the influential stages of the PUF, associated with the influential bits, can be eliminated by rolling the PUF. Finally, we discuss how the Fourier analysis-based attack introduced in [GTS18] can be launched against the PUFs with enhanced hardness.

### 6.1 k-junta Testing

As the first step in a systematic evaluation of the impact of rolling a PUF, we examine if the PUF and its respective rolled PUFs can be represented by $k$-junta functions (see section 2). For that matter, several vital observations should be made. First, the fact that an LTF can model an Arbiter PUF is of the essence of our methodology [GLC+04, MKP08]. Based upon this, it is straightforward to conclude that the noise sensitivity of $f_{PUF}$ is a bounded, small value depending only on $\varepsilon$, namely we have $NS_\varepsilon(f) \leq 8.54\sqrt{\varepsilon}$ [KOS02], cf. [GTS18]. Although this is an interesting result, it does not reflect how close our function $f_{PUF}$ is to a $k$-junta. To show this, we rely on Bourgain's theorem [Bou02], and a tight quantitative statement proved in a work of Kindler and O'Donnell [KO12]. These theorems state that if $NS_\varepsilon(f) = O(\delta\sqrt{\varepsilon})$, $f$ is $\delta$-close to a $p(1/\varepsilon, 1/\delta)$-junta, for some polynomial $p(\cdot)$, and constant and sufficiently small value of $\varepsilon$ and $\delta$. More interesting from the point of view of our work, if $f$ is an LTF and $\mathrm{NS}_\varepsilon(f) = O(\delta^{2-\varepsilon/1-\varepsilon}\sqrt{\varepsilon})$, then the LTF $f$ is $\delta$-close to an $O((1/\varepsilon^2) \cdot \log(1/\varepsilon) \cdot \log(1/\delta))$-junta [DJS+15]. This provides a basis for developing *property testing* algorithms, as presented in, e.g., [MORS10].

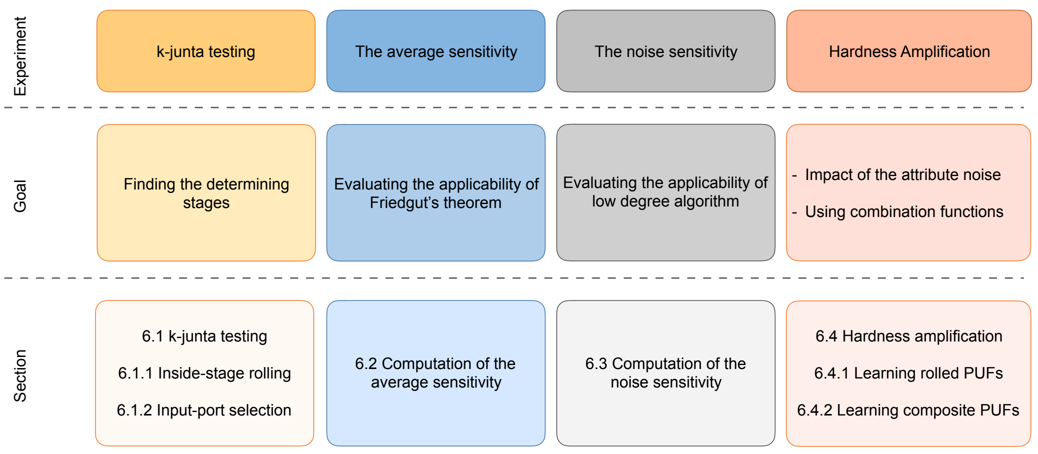
Experiment
k-junta testing
The average sensitivity
The noise sensitivity
Hardness Amplification
Goal
Finding the determining stages
Evaluating the applicability of Friedgut's theorem
Evaluating the applicability of low degree algorithm
- Impact of the attribute noise
- Using combination functions
Section
6.1 k-junta testing
6.1.1 Inside-stage rolling
6.1.2 Input-port selection
6.2 Computation of the average sensitivity
6.3 Computation of the noise sensitivity
6.4 Hardness amplification
6.4.1 Learning rolled PUFs
6.4.2 Learning composite PUFs

Figure 8: Organization of Section 6. The first row presents the types of the experiments conducted, whose results are shown in this section. The goals of these experiments are shown in the second row, whereas the third row is related to, in which section the corresponding results can be found.

The rationale behind a property testing algorithm is that giving black-box query access to an unknown function $f$, and the algorithm determines whether $f$ has a specific, pre-defined property or it is "far" from any function having that property [GGR98]. In view of our approach, we are mainly interested in the fact that property testing algorithms offer insight into approximation problems. Hence, instead of approximating the noise sensitivity of our targeted function $f_{PUF}$, we apply a $k$-junta testing algorithm to examine if it is close to a $k$-junta function. In this regard, in Matlab [Inc] we implement the $k$-junta testing algorithm proposed in [MORS10]. Note that although the scope of [MORS10] mainly focuses on halfspace testing, it covers a useful procedure to test if an unknown function is close to a $k$-junta function.

In a first step, we run the algorithm against our unrolled Arbiter PUF, i.e., a PUF with an inside-LUT configuration AB. In our experiments, we fix $\delta = 0.99$. The value of $\varepsilon$ should be chosen carefully to fulfill requirements enforced by the spectral properties of a Boolean function, e.g., Parseval's identity cf. [MORS10]. With regard to this requirement, we compute an interval, within which $\varepsilon$ can lie. The mean of the values lying within this interval is chosen as the default value of $\varepsilon$ that is, $\varepsilon = 0.0024$. Note that property testing algorithms are by nature statistical, and therefore, in our experiments, we may slightly vary the value of $\varepsilon$ to guarantee the correctness of the output of the algorithm. Moreover, we repeat each experiment 5 times and take the maximum number of influential stages, i.e., the stages associated with the Boolean variable determining the output of the function $f_{PUF}$.

### 6.1.1 Inside-LUT Configuration as a Rolling Strategy

As can be seen in the first row of Table 2, as expected, the function $f_{PUF}$ depends solely on 4 influential stages, namely, the stages (5),(9),(17), and (22) (hereafter, the numbers inside the parentheses indicate the rolled stages). In other rows of Table 2, we present the number of the influential stages, when different LUT configurations are chosen individually for each influential stage.

The results presented in Table 2 are prime examples of what can be understood from conducting a systematic, and mathematically precise study of which LUT configuration should be chosen to prevent strong $k$-junta learning-based attacks, as proposed in [GTFS16,

| LUT Config. | Reconfig. stage | # Influential stages | Influential stages |
|---|---|---|---|
| Unrolled (AB) | – | 4 | 5,9,17,22 |
| AC | 5 | 4 | 5,9,17,22 |
| | 9 | 7 | 7, 9, 5, 17, 47, 32, 22 |
| | 17 | 3 | 5,9,22 |
| | 22 | 1 | 9 |
| AD | 5 | 2 | 9,22 |
| | 9 | 11 | 20,17,3,10,36,22,5,47,9,7,32 |
| | 17 | 3 | 9,22,5 |
| | 22 | 5 | 32,5,9,17,22 |
| BC | 5 | 1 | 9 |
| | 9 | 11 | 17,20,10,22,3,36,47,7,9,5,32 |
| | 17 | - | - |
| | 22 | - | - |
| BD | 5 | - | - |
| | 9 | 9 | 22,47,20,7,9,17,32,3,5 |
| | 17 | - | - |
| | 22 | 3 | 9,17,5 |
| CD | 5 | - | - |
| | 9 | - | - |
| | 17 | 19 | 20,6,7,18,17,5,35,32,47,52,54, 9,50,36,3,22,31,53,4 |
| | 22 | 1 | 9 |

Table 2: Influential stages within $n = 64$ stages for various LUT configurations. In this experiment, $\delta = 0.99$, $\varepsilon = 0.0024$, and each influential stage is reconfigured individually.

GTFS17]. In this context, our methodology gives a firm basis for designing more robust PUFs. From all those interesting results, the following conclusions can be drawn. There are some ineffective inside-LUT rolling strategies, e.g., changing the configuration from AB to the AD configuration in the stage (22) and the AC in the stage (5) do not impair the effect of these influential bits.

Moreover, by changing the inside-LUT configuration, the attacker has to collect more CRPs to find the influential bits. This is due to the fact that for $\varepsilon$ set by us in most of the scenarios (i.e., different configurations of different influential stages), the influential stage could not be found by the algorithm. Hence, the attacker has to expand the window of the Fourier spectrum to identify the whole set of influential bits, and consequently, learn the challenge-response behavior of the PUF. To this end, more CRPs should be collected by the attacker, although even this cannot always be helpful. As an example, consider the stage (9) that is configured by choosing AD option, we expand the Fourier spectrum by running the experiment on 122764 CRPs, about an 8% increase in the number of CRPs. In this case, the algorithm determines the stages (22) and (17) as influential and fails to find other stages that it could figure out before (see the second row of Table 2). Therefore, in this case, applying more CRPs can be even drastically confusing for the attacker. What can be seen in those scenarios 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 no or only some of the influential stages. In addition to the above discussion, one can observe that the BC configuration can be effective under some particular circumstances, e.g., rolling the stages (5), (17), and (22). Among those, rolling the (17) stage can be very useful. Finally, comparing the results for all possible inside-LUT configurations shows that the configuration CD can be one of the best options to remove the effect of influential stages. Hence, in our next experiment, we take into account the stages with CD configuration.

In the next step, we consider the impact of reconfiguring more than one stage at the same time. Table 3 presents the results for combinations, for which the influential stages can be still found after the rolling process. This means that for the combination (17,22), (5,9), (5,22), and all the sets of three and four stages, which are not shown in the table, no influential stage can be found after running the algorithm. With respect to the results presented in Table 3, when rolling the PUF by changing the configuration of a set of

| Reconfig. stages | $\varepsilon$ | # Influential stages | Influential stages |
|---|---|---|---|
| 9, 22 | 0.0024 | 7 | 32,5,17,47,3,20,7 |
| | 0.0023 | 2 | 5,17 |
| 9, 17 | 0.0024 | 16 | 47,18,32,20,7,5,22,35,6, 54,31,3,53,10,50,4 |
| | 0.0023 | 5 | 22,20,32,3,47 |
| 5, 17 | 0.0024 | 2 | 9,22 |
| | 0.0023 | 2 | 9,22 |

Table 3: Influential stages within $n = 64$ stages for various LUT configurations. In this experiment, $\delta = 0.99$, the inside-LUT configuration is CD, and a set of stages is reconfigured.

| Stage | $\varepsilon$ | Input port | # Influential stages | Influential stages |
|---|---|---|---|---|
| 5 | 0.0024 | PDL-I1 | – | – |
| | | PDL-I2 | 1 | 9 |
| | | PDL-I3 | 3 | 17,22,9 |
| | | PDL-I4 | 2 | 22,9 |
| 9 | 0.0022 | PDL-I1 | 2 | 22,5 |
| | | PDL-I2 | 14 | 47,63,40,36,10, 3,17, 56,46,19,18,32,7,4 |
| | | PDL-I3 | – | – |
| | | PDL-I4 | 3 | 5,22,17 |
| 17 | 0.0022 | PDL-I1 | 1 | 9 |
| | | PDL-I2 | 2 | 5,22 |
| | | PDL-I3 | 1 | 9 |
| | | PDL-I4 | 2 | 9,22 |
| 22 | 0.0022 | PDL-I1 | 3 | 22,5,9 |
| | | PDL-I2 | 3 | 17,9,5 |
| | | PDL-I3 | 1 | 9 |
| | | PDL-I4 | – | – |

Table 4: Influential stages within $n = 64$ stages for different input ports. In this experiment, $\delta = 0.99$, the LUT reconfiguration is CD.

stages, the traces of the remaining influential stages can be seen in some experiments. Among those set, two cases can be distinguished; first, for a given level of $\varepsilon$, e.g., $\varepsilon$=0.0024, the adversary can be confused by the number of influential stages determined by the algorithm, see, e.g., the combination (9,22). However, if the number of CRPs available to the adversary is increased, the $\varepsilon$ can be reduced, e.g., to 0.0023, and the correct number of influential stages can be disclosed. Secondly, even after increasing the number of available CRPs, the exact number of the influential stages cannot be determined. For instance, see the combination (9,17), for which the attacker cannot find the exact set of influential stages even after increasing the number of CRPs, and consequently, decreasing $\varepsilon$.

### 6.1.2 Input-port Selection as a Rolling Strategy

Up until this point, we discuss the results achieved by changing the LUT configuration of the influential stages. Last but not least, we demonstrate how choosing different ports for the input signal can be helpful to build more robust PUFs. In some experiments, the value of $\varepsilon$ has been changed for experiments on each stage. We reduce this value from 0.0024 to a lower value as far as the algorithm can find an influential stage.

The key message to be conveyed here is that in principle, changing the input port can be another possibility for rolling the PUF. In a nutshell, for the same value of $\varepsilon$, the previous approach has been more effective, although the influential stages can be removed by taking this approach as well. For that matter, setting the input port of the stage (9) to PDL-I2 is one of the critical observation, where the attacker can be completely fooled; however, it is not always the case for other stages. More importantly, as can be seen in the Table 4, in order to determine the influential stages, the value of $\varepsilon$ should be lowered, in comparison to previous experiments, in which the LUT configuration has been changed.

| Reconfig. stage(s) | Config. strategy | Avg. sensitivity |
|---|---|---|
| – | Unrolled (AB) | 5.95 |
| 5 | CD | 5.85 |
| 9 | CD | 5.88 |
| 17 | CD | 5.60 |
| 22 | CD | 5.91 |
| 5, 9 | CD | 5.81 |
| 9, 17 | CD | 5.40 |
| 5,9,17, 22 | CD | 5.60 |
| 9 | PDL-I2 | 4.83 |
| 22 | PDL-I1 | 5.39 |
| 22 | PDL-I2 | 5.61 |
| 22 | PDL-I3 | 5.70 |
| 22 | PDL-I4 | 5.92 |

Table 5: The average sensitivity of Arbiter PUFs with $n = 64$ stages and different rolling strategies.

In other words, when rolling the PUF by modifying the input port, (only) in some cases, more CRPs are required to figure out which stage is influential.

To sum up this section, we stress that in order to make a PUF more robust to machine learning attacks depending on finding the influential stages, one has to go step-by-step and identify the best strategy to roll the PUF.

## 6.2 Computation of the Average Sensitivity

In addition to the $k$-junta testing approach, [GTFS16] has suggested that according to Friedgut's theorem [Fri98], it is possible to come up with the total number of influential bits of the function $f_{PUF}$, with $\varepsilon$ accuracy level. To this end, one should compute the average sensitivity of the targeted function, i.e., $f_{PUF}$ in our case. In our experiments, the process of computing this has been borrowed from [RRS+12]. We follow this by calculating the average sensitivity of the reference, unrolled PUF and its corresponding rolled PUFs. Although we compute the average sensitivity of all the PUFs with different configurations, as presented in the previous section (see Section 6.1), here we demonstrate the results achieved for the most interesting instances of the rolled PUF, see Table 5.

The underlying message of these results is that the average sensitivity of $f_{PUF}$ is a constant value ($O(1)$) that varies slightly and insignificantly for each rolling strategy. Even for the case that the stage (9) is rolled by choosing the input-port PDL-I2, the difference cannot be considered statistically significant enough to justify a change in this trend. However, by applying Friedgut's theorem, the value of $k$ cannot be approximated in our case [Fri98, GTFS16]. In more details, when the average sensitivity exceeds the bound $\ln n$, the number of influential variables cannot be calculated according to Friedgut's theorem. Now, the natural question is whether our PUF can be represented by a $k$-junta, with a small $k$. This has been addressed and well formulated by O'Donnell as follows [O'D03]. With regard to the results presented by Friedgut [Fri98], having low average sensitivity could be a sufficient condition for being close to a junta. Nonetheless, this has been shown not to be necessary as there exist functions being close to juntas with a very large average sensitivity. This issue has been addressed by Bourgain's results that show in fact, all functions with sufficiently low noise sensitivity must be close to a junta [Bou02, O'D03] Therefore, to support the results that we have presented in Section 6.1, we compute the noise sensitivity of the functions representing our PUFs.

## 6.3 Computation of the Noise Sensitivity

The importance of the notion of noise sensitivity as a metric for the assessment of the learnability of a PUF has been stressed in [GTS18]. However, in their investigation, the

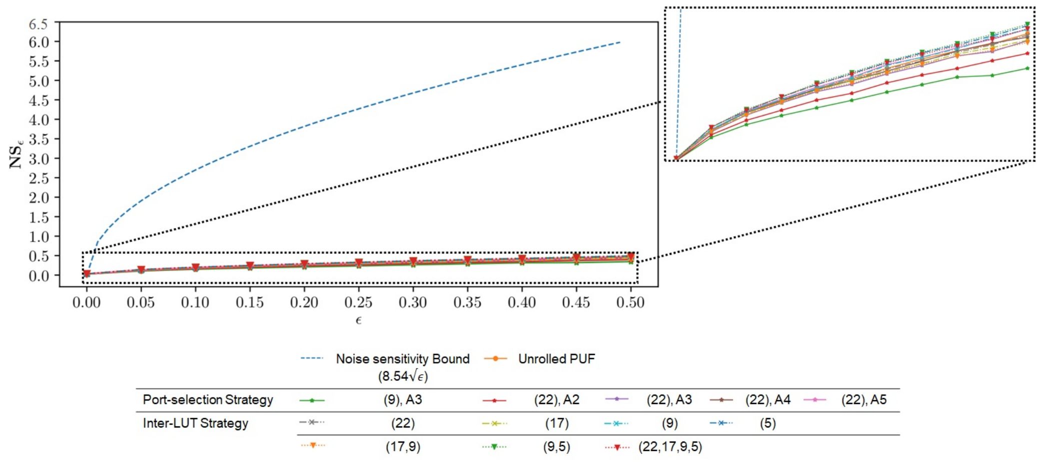

Figure 9: The noise sensitivity of unrolled and rolled PUF following different rolling strategies.

authors of [GTS18] have not computed the average sensitivity of the PUFs, e.g., Arbiter PUFs. Here we approximate the noise sensitivity according to its definition provided in Section 2.1. To this end, well-structured CRPs are required following a certain pattern. First, we uniformly, randomly choose a challenge $c$, and then, generate an $\varepsilon$-noisy version of it, i.e., $c'$. Accordingly, we generate 50000 uniformly, randomly chosen strings and 50000 noisy versions of these so that we get a total number of 100000 challenges for a particular $\varepsilon$. This procedure is repeated for different $\varepsilon \in [0, 0.5]$. Figure 9 shows the noise sensitivity for some rolling options, which are interesting according to the discussion in Section 6.1. Moreover, Figure 9 depicts the upper bound of the noise sensitivity for LTFs [KOS02]. As can be seen, the noise sensitivity of the unrolled and rolled PUFs is far below the upper bound. More interestingly, the trend of the curves presented for each PUF is the same, namely a square root function. These results confirm the results achieved by applying the $k$-junta tests: PUFs exhibiting a small noise sensitivity can be represented by a $k$-junta. This conclusion leads us directly to the next step, in which we investigate how the robustness of a PUF can be systematically increased.

## 6.4 Hardness Amplification

### 6.4.1 Learning Rolled PUFs

In this section, we present the results related to our strategies for achieving more robust PUFs. First of all, we examine if the rolled PUFs are still vulnerable to machine learning attacks, in particular, applying the low degree algorithm[9]. In our first experiment, we set the noise level ($\varepsilon$) equal to 0.05. The number of CRPs used to come up with an approximator of the Boolean function representing a PUF is 1000, as shown in the first part of the Table 6, whereas in the test set we have 100000 CRPs. What can be understood from the results of this experiment is that the PUFs, which are chosen based on their interesting results of the $k$-junta tests, can be still modeled. We emphasize that the difference between the accuracy of the models delivered for each PUF is not statistically significant. More importantly, even with a much smaller number of CRPs, i.e., 125 as shown in the second part of Table 6, eventually the same results can be achieved.

In line with the explanation given in Section 3.2, it is possible to determine the minimum number of CRPs required to deliver a desirably accurate approximator of the Boolean

[9]We have already explained the reason behind this choice in Section 3.1

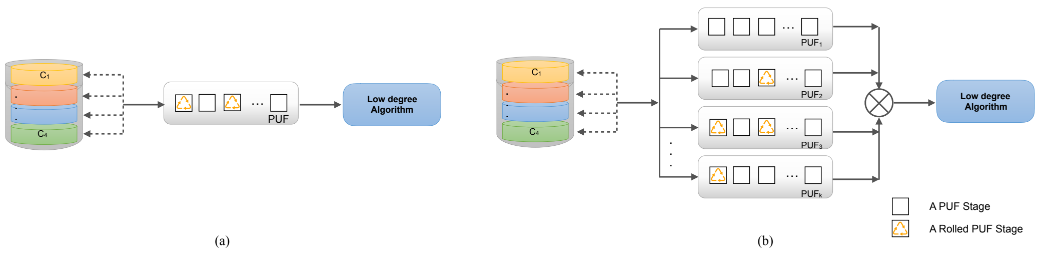

Figure 10: The schematic illustrating how our scheme is used in different rounds of rolling. The PUFs used in our experiments are Arbiter PUFs. Moreover, the combination function shown in this scheme is either the XOR or the Tribes functions. The response of the PUF is given to the low degree algorithm to evaluate the hardness of the PUF.

function representing a PUF. We take into consideration the generic lower bound that holds not only for the low degree algorithm, but also for every PAC learning algorithm. Our observation confirms that when reducing the number of CRPs to 100, 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 are forced to shift our focus to designing rolling strategies changed in each round. Moreover, 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 two strategies for feeding the CRPs into the PUFs in different rolling rounds. The first CRP management strategy is called "static" reflecting the fact that the same CRPs are used in each and every round of rolling. It is to be noted that due to the existence of the attribute noise, this is applicable since the attacker cannot guess if the response to the same challenge could be flipped in each round. As for the "refined" CRP management strategy, the used CRPs are filtered (see Figure 10(a)). In both of the static and the refined methods, the number of CRPs used in a round is 100, and the adversary can take advantage of all the CRPs observed by her. This means that after the $i^{\text{th}}$ round, the number of CRPs in the training set is $100 \cdot i$. Additionally and similarly, the number of CRPs in the test set is $5000 \cdot i$. Table 7 shows the results of the experiments, in which the effectiveness of the CRP management strategies as well as the rolling strategies is examined. As can be seen in this table, the accuracy of the model is reduced drastically to approximately 65%. Albeit the differences in different rolling and CRP management strategies, no statistically-relevant difference between them can be observed.

### 6.4.2 Learning Composite PUFs

The goal of conducting this set of experiments is to evaluate to what extent the hardness of a PUF can be amplified through two specific strategies; first, combining some chains of the Arbiter PUFs, and secondly, by rolling these PUFs. We begin with a composite PUF 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

| Stage | # CRPs in training set | Accuracy |
|---|---|---|
| Unrolled (AB) | 1000 | 94.75 |
| (5) | | 97.13 |
| (9) | | 93.88 |
| (17) | | 92.87 |
| (22) | | 94.26 |
| (5,9) | | 96.79 |
| (9,17) | | 97.18 |
| (17,22) | | 94.66 |
| (5,9,17,22) | | 97.81 |
| Unrolled (AB) | 125 | 94.26 |
| (5) | | 96.79 |
| (9) | | 92.96 |
| (17) | | 94.75 |
| (22) | | 93.66 |
| (5,9) | | 93.18 |
| (9,17) | | 93.00 |
| (17,22) | | 92.13 |
| (5,9,17,22) | | 95.21 |

Table 6: Results for learning Arbiter PUFs rolled according to different strategies. In all of the experiments, the noise level is set to 0.05, the number of CRPs in the test set is 100000, and the inter-LUT configuration is CD.

| CRP Set | Rolling Strategy | Accuracy [%] | | | |
|---|---|---|---|---|---|
| | | 1 round of rolling | 2 round of rolling | 3 round of rolling | 4 round of rolling |
| Static | 1 | 64.10 | 63.15 | 62.82 | 62.55 |
| | 2 | 64.67 | 64.49 | 63.32 | 63.24 |
| Refined | 1 | 65.26 | 64.16 | 65.02 | 64.05 |
| | 2 | 65.35 | 64.30 | 65 | 64.09 |

Table 7: 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$\rightarrow$5$\rightarrow$9$\rightarrow$17$\rightarrow$22, whereas strategy 2 is related to Unrolled$\rightarrow$(5,9)$\rightarrow$(9,17) $\rightarrow$(17,22)$\rightarrow$ (5,9,17,22).

length of the Tribes functions, $k$, to 4, 6, and 12 accordingly. Moreover, as suggested in [O'D04], $b$, the size of the terms in the Tribes functions is set to 2 that is close to $\log_2 k$. To roll the PUFs in each round, for each chain of the composite PUFs we first determine the influential stages, see Figure 10(b). Before elaborating on the results, we underline that the results presented for XOR Arbiter PUFs with a large number of chains do not contradict the upper bound established for provable algorithms, cf. [GTS15]. This can be explained given the fact that the XOR Arbiter PUFs discussed in this paper are composed of PUFs that are not entirely independent. Put differently, in the presence of the attribute noise, the responses of the PUFs are correlated. Therefore, the results presented here are not comparable to what has been reported in the literature so far, e.g., [GTS15].

In each round of rolling, at least one of the influential stages, determined in Section 6.1, is reconfigured. The number of CRPs in training and test sets are the same as those in the experiment on rolled PUFs (see Section 6.4.1). Specifically, we allow the adversary to observe 100 CRPs during each round of our scheme, and to collect these CRPs as well as to run the low degree algorithm on the set collected by her after each round (see Figure 10(b)). In a similar fashion, the number of CRPs in the test set is $5000 \cdot i$. As mentioned before, the responses corresponding to these challenges are measured 19 times, and by applying the majority voting technique, we resolve the noisy CRPs.

Table 8 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 Section 6.4.1 for the corresponding results). The theoretically precise extent of this improvement can be estimated. Roughly speaking, the hardness of the combined functions is at most equal the expected bias of the combination function.

Besides, when it is impossible to compute the expected bias, e.g., for the Tribes function, it can be well-estimated by the noise stability or, equivalently, the noise sensitivity of that function. For Tribes functions, the noise stability[10] is extremely small, especially, when $\delta$ is close to $1/2$, it is as small as $1/2 + 1/k$. For the XOR function, the expected bias can be calculated as $1/2 + 1/2(1-\delta)^k$, where $\delta$ here denotes the hardness of the function $f_{\mathrm{PUF}}$. According to the results provided in Section 6.4.1, on average, we can set $\delta$ 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 noise stability of our Tribes function can be approximately 0.75 and 0.6667, respectively. Consequently, the expected bias and the hardness of them can be at least equal these values.

Nevertheless, as can be seen in Table 8, for the XOR function, the combined function may not be as hard as what can be approximated theoretically. This can be seen as a result of a key difference between our practical setting and the theoretical one. In our setting the functions $f_{\mathrm{PUF}_1}, \cdots, f_{\mathrm{PUF}_k}$ are independent, but according to the rolling procedure and its relationship to the attribute noise, the challenges applied to the PUFs are "corrupted". Intuitively, if the adversary comes up with the response of a PUF to a challenge, e.g., $f_{\mathrm{PUF}_1}(c_i)$, with the probability $1 - \mathrm{NS}_{\varepsilon}(f_{\mathrm{PUF}})$ the response of other PUFs to $c_i$ can be equal to $f_{\mathrm{PUF}_1}(c_i)$. But, the adversary does not *know* whether the response of the PUF $f_{\mathrm{PUF}_1}(c_i)$ itself is corrupted. If the adversary gains this knowledge, the above situation is called the *hard-core* scenario [O'D04, Imp95]. The hardness amplification bounds provided for the XOR and the Tribes functions are established in the hard-core setting. However, we go beyond this and assume the worst case scenario, where the attacker does not know if the response of the PUF $f_{\mathrm{PUF}_1}(c_i)$ is corrupted. Consequently, there is no one-to-one comparison between the bounds suggested in [O'D04] and our results. Therefore, when XORing the functions $f_{\mathrm{PUF}_1}, \cdots, f_{\mathrm{PUF}_k}$, our composite 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 affecting the inputs (for more details, see [O'D04]). Hence, the Tribe functions can handle this matter, when conducting experiments on real-world PUFs. For the composite PUFs affected by the attribute noise, for instance, combined by using XOR (12) or Tribes (12,3), we calculate the expected bias. As for the Tribes (12,3), the expected bias is 0.5178, and it is 0.7689 for XOR (12). In conclusion, although one cannot directly compare the results shown in Table 8 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.

Another interesting aspect of the results is related to how the hardness of the composite 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.

# 7 Discussion

## 7.1 Theoretical Considerations

**Comparison to PUF Protocols:** First and foremost, we emphasize that the scheme proposed in this paper should not be confused with PUF protocols as suggested in, e.g., [RMK+14, YHD+16]. Nevertheless, our scheme shares some notable commonalities with such approaches. First of all, the primary goal of our approach and the protocols

[10] We put emphasis again on the fact that here our focus is mainly on the attribute noise.

| Combination Function | Accuracy [%] | | | |
|---|---|---|---|---|
| | 1 round of rolling | 2 round of rolling | 3 round of rolling | 4 round 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 8: 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.

mentioned above is to stop an attacker launching an ML attack; however, as shown for the Slender PUF protocol [DPGV15], this goal cannot always be attained. More precisely, as formulated in a suitable fashion in [Del17], being resistant to ML attacks should not be connected to lower bounds established for some specific ML algorithms. 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, which has been based upon that lower/upper bounds. Hence, we should come up with bounds that hold in general, i.e., for any provable algorithm, see, e.g., [GTFS17, GTFS16]. In contrast to protocols discussed above, our scheme is anchored on a solid foundation: the hardness amplification for Boolean functions.

The second common aspect of our scheme and the PUF-based protocols is applying the notion of "controlled PUFs". As an example, the Slender PUF protocol [RMK+14] deploys an input network to not only restrict the access of the adversary to the CRPs, but also meet the SAC requirement. In our scheme, on the contrary, 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.

**Lower Bound on the CRPs for Learning Tribes Functions:** In general, lower bounds are established to provide an answer to the question of how much resources (i.e., time, examples, etc.) are required to learn a target concept. We have already discussed two possible approaches to establish such lower bounds in Section 3.2. Here to establish a lower bound for Tribes-combined Arbiter PUFs, we consider both of those approaches, namely the general lower bound and the lower bound for the low degree algorithm. Let us begin with the general lower bound, where it suffices to calculate the Vapnik-Chervonenkis dimension of the composite function $C(f_{\text{PUF}}(x_1), \cdots, f_{\text{PUF}}(x_k))$, where $C(\cdot)$ is the Tribes function. As for the representation of the target concept, $f_{\text{PUF}}$, we stick to the LTF functions. It is known that for finite concept classes, $\text{VC}_{\text{dim}}(C \otimes F) = O(kd + kd\log(kd))$, where $\otimes$ denotes the composition operator and $d = \text{VC}_{\text{dim}}(F)$ [Sha09]. Now when the inputs to this Tribes function are LTFs with $\text{VC}_{\text{dim}}(F) = n + 1$, we obtain

$$\text{VC}_{\text{dim}}(C(f_{\text{PUF}}(x_1), \cdots, f_{\text{PUF}}(x_k))) = O(k(n+1)(1 + \log(kn + k))).$$

The key message that this equation conveys is that the Vapnik-Chervonenkis dimension of our composite function is increased, when increasing $k$ and $n$. Now substituting the above dimension in the general lower bound (see Section 2.2) yields the minimum number of CRPs required to learn our composite function. Adopting the same approach for the XOR combination function results in the same bound.

As explained before, the above lower bound holds in general, i.e., it is not specific to any provable algorithm, unlike bounds established for a pre-defined algorithm. Yet, for the sake of completion, here we provide the lower bound for the low degree algorithm that is tighter towards the respective upper bound. In an attempt to compute this bound, we begin with the following fact. The noise sensitivity of *any* composite function of LTFs defined over $k$ LTFs is $\mathrm{NS}_\varepsilon \leq O(k\sqrt{\varepsilon})$. It is straightforward to show that the lower bound provided in Section 3.2 should be slightly modified so that $d = k^2/\varepsilon$. In other words, contrary to the general lower bound discussed above, one can observe that an increase in $k$ results in an exponential growth in the minimum number of CRPs required to learn our composite function (see the lower bound presented in Section 3.2).

It is crucial to recall that the above lower bounds serve to provide a better insight into the number of CRPs in each round, which is necessary and sufficient to obtain a $1-\delta$-hard PUF. And 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, composite function, cf. Section 2 in [O'D04]. As the expected bias of the Tribes functions in the presence of the attribute noise is close to one half, in contrast to the XOR function, the hardness of the composite function can be guaranteed.

**Note on the Applicability of Boosting Techniques:** The natural question that should be answered here is whether boosting or, more generally, ensemble methods can be applied to learn our proposed scheme. This question is valid while one can observe that the setting of the ML algorithm applied against our scheme can be considered as the setting for a weak PAC learner (see Section 3.2). The main factor differentiating a weak learner from its analog, i.e., strong PAC learner, is the level of the accuracy that is achievable for the same number of examples and the level of confidence. More specifically, a weak learner can achieve only slightly better than flipping a coin [SF12]. Although such learner could seem inappropriate, ensemble methods are introduced to make use of a weak learner and transform it into a strong learner.

The key idea behind an ensemble method is to include models obtained after every round of learning in an ensemble and combine them to classify a new, unseen example [Die00]. A prime example of ensemble methods is boosting techniques attempting to minimize a particular error function step-by-step. Although for some boosting algorithm the presence of noise results in overfitting, there exist boosting techniques designed specifically to handle the issue with the noisy examples. In this regard, various attempts made to enhance boosting techniques, however, the majority of them take into account the case of classification noise. In fact, the case of examples featuring the attribute noise is difficult to address and does not receive much attention. Among a few positive results achieved in this matter, one of the most celebrated boosters is introduced in [Ser03], namely, SmoothBoost. SmoothBoost addresses the issue with the attribute noise thanks to the fact that it does not generate skewed distributions by not putting a great deal of weight on noisy examples, in contrast to other boosters, e.g., AdaBoost. Although not explicitly mentioned, the bound on the noise that SmoothBoost can tolerate depends heavily on the distribution of the examples and the number of Boolean variables involved in the function. More precisely, if the examples are drawn uniformly, the attribute noise rate must be far less than $\varepsilon^2/\log(n/\varepsilon)$ [KLS09]. To recap, in order to ensure that such booster fails, the underlying parameters related to the attribute noise and how the examples are drawn in each round should be considered carefully. As an example, for a 64-bit Arbiter PUF with $\varepsilon$=0.51, if the noise level exceeds 2.5%, it is impossible to apply SmoothBooster against our scheme. According to the results of experiments that we have conducted to estimate the attribute noise rate, irrespective of the rolling strategy, it exceeds the above bound for our rolling PUFs.

**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 leveraging the impact of the classification noise either for attacking a PUF [Bec15, DV14] or for improving its security [YVDM14]. The latter approach is interesting; however, it has been shown to be less effective as promised [TB15]. Attacks that take advantages of the information disclosed by the classification noise, similar to side channel information, mainly rely on the meta-stability condition at the measuring element of PUFs (i.e., the frequency comparators, and the arbiter). In our scheme, we resolve this by applying a known and widely-accepted technique, namely the majority voting. Consequently, the attacker has to deal with the attribute noise primarily, which could impair the effectiveness of the attacks suggested in [Bec15, DV14]. More precisely, the equations that have to be solved to build a model of a PUF (cf. Equation 4b in [Bec15] and the system of linear equations in Section VI.A and Section VI.B in [DV14]) cannot be formulated as required, due to the presence of the attribute noise.

## 7.2 Practical Considerations

**Existence of the Attribute Noise in Real-world Implementations:** One could argue that by increasing the number of PUF chains XORed together, it can be possible to achieve the same hardness level as obtained for the Tribes functions. We should underline the requirement that the composite PUF should remain close to balanced, i.e., with a low level of bias, when combining some PUF chains with this property. The importance lies in the fact 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. It is also interesting to observe that for a composite PUF involving the XOR combination function, the level of the attribute noise may not reach the level that can be achieved by rolling the PUF stages. In this case, the bounds established for the provable algorithm (see, e.g., [GTS15]) can be further applied. However, if the level of the attribute noise is sufficiently high to influence the bias of the PUF chains, the desired hardness cannot be achieved, even for a large number of chains XORed together. Accordingly, as proved in theory and verified in practice, meeting the close-to-balanced condition can be challenging. 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 composite PUF, such as XOR PUF, might bias the overall responses of the composite 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.

**Integrity of PUF Configurations and Challenges:** Although our framework in this paper addresses mainly the passive ML attacks, and not side-channel and physical attacks, we briefly review the potential threats from a more sophisticated adversary, who can mount side-channel/physical attacks. As described in section 5, one approach to realize rolling PUFs is through utilizing the available dynamic reconfiguration feature of FPGAs. In this case, the PUF configuration of each rolled PUF is sent to the FPGA with partial bitstreams. The challenges of the PUF can also be included in the partial bitstream along with the configuration of each rolled PUF, however, they can be fed into the PUF after each reconfiguration via I/O pins.

If the adversary can tamper with the partial bitstreams, she can modify the PUF configuration. As a result, she can connect the output of the PUF directly to one of the I/O pins of the chip and directly read out the raw response of each individual PUF prior to the combining function. Similarly, she can tamper with the challenges, and apply arbitrarily challenges to the PUF. In both cases, the confidentiality and integrity of the PUF have to be guaranteed. To assure confidentiality and integrity of the partial bitstreams, one can deploy the bitstream encryption and authentication methods, provided as features on the latest modern FPGAs. Since the security and vulnerabilities of FPGA IP protection schemes have been well studied during the last decade, the latest integrated countermeasures offered by FPGA vendors avert several classes of side-channel and physical attacks. However, using encryption and authentication to secure the PUF might be considered as on overhead. While activation of bitstream encryption and authentication is optional on SRAM-based FPGAs (e.g., Xilinx and Intel FPGAs), these features are always activated on Flash-based FPGAs (e.g., Microsemi FPGAs). In any case, we stress that the PUF circuit is only one of the several IP cores, which is implemented on the FPGA, and the confidentiality and integrity of other IP cores has to be ensured as well. Therefore, it is realistic to assume that the bitstream encryption and authentication is already in use by the user.

**Enrollment Phase:** Since the influential stages can be different from a device to another, the framework presented in Section 6 has to be applied to each new PUF during the enrollment phase. Therefore, in addition to CRPs, the configuration of each rolled PUF as well as a chip ID (e.g., a device DNA) have to be stored in a database for the future authentications. Note that the device ID is not required to be kept secret, as it does not divulge any extra information regarding the behavior of the PUF.

## 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 extensive discussion on methods previously 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 observation, it has been suggested to roll a PUF so that CRPs collected by an attacker are generated from several instances of the PUF, and therefore, the ML attack could fail. On the other hand, the XOR function has been introduced to obfuscate the responses of PUFs combined together. Our proposed scheme takes advantage of both of these techniques, namely, an increase in the attribute noise and combining some 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, being known to the ML community, 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 extensive experiments on real-world PUFs, whose results are presented in this paper.

## References

[Ach18] Achronix Semiconductor Corporation. Achronix Product Brief. https://www.achronix.com/wp-content/uploads/2017/05/Speedcore_eFPGA_Product_BriefPB028.pdf, 2018.

[AL88] Dana Angluin and Philip Laird. Learning from Noisy Examples. *Machine Learning*, 2(4):343–370, 1988.

[AMS$^+$11] Frederik Armknecht, Roel Maes, A Sadeghi, O-X Standaert, and Christian Wachsmann. A Formalization of the Security Features of Physical Functions. In *Security and Privacy (SP), 2011 IEEE Symp. on*, pages 397–412, 2011.

[AMSY16] Frederik Armknecht, Daisuke Moriyama, Ahmad-Reza Sadeghi, and Moti Yung. Towards a Unified Security Model for Physically Unclonable Functions. In *Topics in Cryptology-CT-RSA 2016: The Cryptographers' Track at the RSA Conf.*, volume 9610, page 271. Springer, 2016.

[Bec15] Georg T Becker. The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs. In *Cryptographic Hardware and Embedded Systems–CHES 2015*, pages 535–555. Springer, 2015.

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the Vapnik-Chervonenkis Dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[BJT03] Nader H Bshouty, Jeffrey C Jackson, and Christino Tamon. Uniform-Distribution Attribute Noise Learnability. *Information and Computation*, 187(2):277–290, 2003.

[BOL89] Michael Ben-Or and Nathan Linial. Collective Coin Flipping. *Advances in Computing Research*, 5:91–115, 1989.

[Bou02] Jean Bourgain. On the Distribution of the Fourier Spectrum of Boolean Functions. *Israel Journal of Mathematics*, 131(1):269–276, 2002.

[BS17] Armin Babaei and Gregor Schiele. Spatial Reconfigurable Physical Unclonable Functions for the Internet of Things. In *Intl. Conf. on Security, Privacy and Anonymity in Computation, Communication and Storage*, pages 312–321. Springer, 2017.

[Del17] Jeroen Delvaux. *Security Analysis of PUF-based Key Generation and Entity Authentication.* PhD thesis, Ph. D. dissertation, Shanghai Jiao Tong University, China, 2017.

[Die00] Thomas G Dietterich. Ensemble Methods in Machine Learning. In *Intl. workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[DJMW12] Yevgeniy Dodis, Abhishek Jain, Tal Moran, and Daniel Wichs. Counterexamples to Hardness Amplification Beyond Negligible. In *Theory of Cryptography Conf.*, pages 476–493. Springer, 2012.

[DJS$^+$15] Ilias Diakonikolas, Ragesh Jaiswal, Rocco A Servedio, Li-Yang Tan, and Andrew Wan. Noise Stable Halfspaces Are Close to Very Small Juntas. *Chicago Journal OF Theoretical Computer Science*, 4:1–13, 2015.

[DLSS13] Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. From Average Case Complexity to Improper Learning Complexity. *arXiv preprint arXiv:1311.2272*, 2013.

[DPGV15] Jeroen Delvaux, Roel Peeters, Dawu Gu, and Ingrid Verbauwhede. A Survey on Lightweight Entity Authentication with Strong PUFs. *ACM Computing Surveys (CSUR)*, 48(2):26, 2015.

[DV14] Jeroen Delvaux and Ingrid Verbauwhede. Fault Injection Modeling Attacks on 65 nm Arbiter and RO Sum PUFs via Environmental Changes. *Circuits and Systems I: Regular Papers, IEEE Trans. on*, 61(6):1701–1713, 2014.

[Fle18] Flex Logix Technologies, Inc. Modular eFPGA. https://static1.squarespace.com/static/548d5bd7e4b087dc74cd6e6d/t/5bc26adb0852293bdde73d77/1539468017196/2018+10+Modular+eFPGA+101018+v1p0.pdf, 2018.

[FLS11] Vitaly Feldman, Homin K Lee, and Rocco A Servedio. Lower Bounds and Hardness Amplification for Learning Shallow Monotone Formulas. In *Proc. of the 24th Annual Conf. on Learning Theory*, pages 273–292, 2011.

[Fri98] Ehud Friedgut. Boolean Functions with Low Average Sensitivity Depend on Few Coordinates. *Combinatorica*, 18(1):27–35, 1998.

[GCVDD02] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Controlled Physical Random Functions. In *Comp. Security Applications Conf., 2002. Proc.. 18th Annual*, pages 149–160, 2002.

[GGR98] Oded Goldreich, Shari Goldwasser, and Dana Ron. Property Testing and Its Connection to Learning and Approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

[GLC$^+$04] Blaise Gassend, Daihyun Lim, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Identification and Authentication of Integrated Circuits. *Concurrency and Computation: Practice and Experience*, 16(11):1077–1098, 2004.

[GNW11] Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR-Lemma. *Studies in Complexity and Cryptography*, 6650:273–301, 2011.

[GS95] Sally A. Goldman and Robert H. Sloan. Can PAC Learning Algorithms Tolerate Random Attribute Noise? *Algorithmica*, 14(1):70–84, 1995.

[GS15] Stefan Gehrer and Georg Sigl. Using the Reconfigurability of Modern FPGAs for Highly Efficient PUF-based Key Generation. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015 10th Intl. Symp. on*, pages 1–6. IEEE, 2015.

[GTFS16] Fatemeh Ganji, Shahin Tajik, Fabian Fäßler, and Jean-Pierre Seifert. Strong Machine Learning Attack against PUFs with No Mathematical Model. In *Intl. Conf. on Cryptographic Hardware and Embedded Systems*, pages 391–411, 2016.

[GTFS17] Fatemeh Ganji, Shahin Tajik, Fabian Fäßler, and Jean-Pierre Seifert. Having No Mathematical Model May Not Secure PUFs. *Journal of Cryptographic Engineering*, 2017.

[GTS15] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. Why Attackers Win: On the Learnability of XOR Arbiter PUFs. In *Trust and Trustworthy Computing*, pages 22–39. Springer, 2015.

[GTS18] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. A Fourier Analysis Based Attack against Physically Unclonable Functions. In *Intl. Conf. on Financial Cryptography and Data Security*. Springer, 2018.

[HVV06] Alexander Healy, Salil Vadhan, and Emanuele Viola. Using Nondeterminism to Amplify Hardness. *SIAM Journal on Computing*, 35(4):903–931, 2006.

[Imp95] Russell Impagliazzo. Hard-core Distributions for Somewhat Hard Problems. In *Foundations of Computer Science, 1995. Proc., 36th Annual Symp. on*, pages 538–545. IEEE, 1995.

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

[KLS09] Adam R Klivans, Philip M Long, and Rocco A Servedio. Learning Halfspaces with Malicious Noise. *Journal of Machine Learning Research*, 10(Dec):2715–2740, 2009.

[KO12] Guy Kindler and Ryan O'Donnell. Gaussian Noise Sensitivity and Fourier Tails. In *Computational Complexity (CCC), 2012 IEEE 27th Annual Conf. on*, pages 137–147. IEEE, 2012.

[KOS02] Adam R Klivans, Ryan O'Donnell, and Rocco A Servedio. Learning Intersections and Thresholds of Halfspaces. In *Foundations of Computer Science, 2002. Proc. The 43rd Annual IEEE Symp. on*, pages 177–186, 2002.

[KV94a] Michael Kearns and Leslie Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[KV94b] Michael J Kearns and Umesh Virkumar Vazirani. *An Introduction to Computational Learning Theory.* MIT press, 1994.

[Lev87] Leonid A Levin. One Way Functions and Pseudorandom Generators. *Combinatorica*, 7(4):357–363, 1987.

[LLG$^+$04] Jae W Lee, Daihyun Lim, Blaise Gassend, G Edward Suh, Marten Van Dijk, and Srini Devadas. A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications. In *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symp. on*, pages 176–179, 2004.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant Depth Circuits, Fourier Transform, and Learnability. *Journal of the ACM (JACM)*, 40(3):607–620, 1993.

[Mae13] Roel Maes. An Accurate Probabilistic Reliability Model for Silicon PUFs. In *Cryptographic Hardware and Embedded Systems-CHES 2013*, pages 73–89. Springer, 2013.

[Man94] Yishay Mansour. Learning Boolean Functions via the Fourier Transform. In *Theoretical Advances in Neural Computation and Learning*, pages 391–424. Springer, 1994.

[MGV08] Nele Mentens, Benedikt Gierlichs, and Ingrid Verbauwhede. Power and Fault Analysis Resistance in Hardware through Dynamic Reconfiguration. In *Intl. Workshop on Cryptographic Hardware and Embedded Systems*, pages 346–362. Springer, 2008.

[MKD10] Mehrdad Majzoobi, Farinaz Koushanfar, and Srinivas Devadas. FPGA PUF Using Programmable Delay lines. In *Information Forensics and Security (WIFS), 2010 IEEE Intl. Workshop on*, pages 1–6, 2010.

[MKP08] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Lightweight Secure PUFs. In *Proc. of the 2008 IEEE/ACM Intl. Conf. on Comp.-Aided Design*, pages 670–673, 2008.

[MKP09] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Techniques for Design and Implementation of Secure Reconfigurable PUFs. *ACM Trans. on Reconfigurable Technology and Systems (TRETS)*, 2, 2009.

[MORS10] Kevin Matulef, Ryan O'Donnell, Ronitt Rubinfeld, and Rocco A Servedio. Testing Halfspaces. *SIAM Journal on Computing*, 39(5):2004–2047, 2010.

[O'D03] Ryan William O'Donnell. *Computational Applications of Noise Sensitivity*. PhD thesis, Massachusetts Institute of Technology, 2003.

[O'D04] Ryan O'Donnell. Hardness Amplification within NP. *Journal of Computer and System Sciences*, 69(1):68–94, 2004.

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

[Qui86] J Ross Quinlan. The Effect of Noise on Concept Learning. *Machine learning: An artificial intelligence approach*, 2:149–166, 1986.

[RMK$^+$14] Masoud Rostami, Mehrdad Majzoobi, Farinaz Koushanfar, D Wallach, and Srinivas Devadas. Robust and Reverse-Engineering Resilient PUF Authentication and Key-Exchange by Substring Matching. *Emerging Topics in Computing, IEEE Trans. on*, 2(1):37–49, 2014.

[RRS$^+$12] Dana Ron, Ronitt Rubinfeld, Muli Safra, Alex Samorodnitsky, and Omri Weinstein. Approximating the Influence of Monotone Boolean Functions in $O(\sqrt{n})$ Query Complexity. *ACM Trans. on Computation Theory (TOCT)*, 4(4):11, 2012.

[RSS$^+$10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling Attacks on Physical Unclonable Functions. In *Proc. of the 17th ACM Conf. on Comp. and Communications Security*, pages 237–249, 2010.

[SBP16] Alexander Spenke, Ralph Breithaupt, and Rainer Plaga. An Arbiter PUF Secured by Remote Random Reconfigurations of an FPGA. In *Intl. Conf. on Trust and Trustworthy Computing*, pages 140–158. Springer, 2016.

[SD07] G Edward Suh and Srinivas Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Proc. of the 44th annual Design Automation Conf.*, pages 9–14, 2007.

[Ser03] Rocco A Servedio. Smooth Boosting and Learning with Malicious Noise. *Journal of Machine Learning Research*, 4(Sep):633–648, 2003.

[SF12] Robert E Schapire and Yoav Freund. *Boosting: Foundations and Algorithms*. MIT press, 2012.

[Sha09] Amnon Shashua. Introduction to Machine Learning: Class Notes 67577. *arXiv preprint arXiv:0904.3664*, 2009.

[SMCN18] Durga Prasad Sahoo, Debdeep Mukhopadhyay, Rajat Subhra Chakraborty, and Phuong Ha Nguyen. A Multiplexer-Based Arbiter PUF Composition with Enhanced Reliability and Security. *IEEE Trans. on Computers*, 67(3):403–417, 2018.

[SMMG15] Pascal Sasdrich, Amir Moradi, Oliver Mischke, and Tim Guneysu. Achieving Side-channel Protection with Dynamic Logic Reconfiguration on Modern FPGAs. In *2015 IEEE Intl. Symp. on Hardware Oriented Security and Trust (HOST)*, pages 130–136. IEEE, 2015.

[TB15] Johannes Tobisch and Georg T Becker. On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation. In *Intl. Workshop on Radio Frequency Identification: Security and Privacy Issues*, pages 17–31. Springer, 2015.

[Tre03] Luca Trevisan. List-decoding Using the XOR Lemma. In *Foundations of Computer Science, 2003. Proc. 44th Annual IEEE Symp. on*, pages 126–135. IEEE, 2003.

[VC71] VN Vapnik and A Ya Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and its Applications*, 16(2):264, 1971.

[Xil] Inc. Xilinx. *Vivado Design Suite User Guide.*

[Yao82] Andrew C Yao. Theory and Application of Trapdoor Functions. In *Foundations of Computer Science, 23rd Annual Symp. on*, pages 80–91. IEEE, 1982.

[YGHL17] Jing Ye, Yue Gong, Yu Hu, and Xiaowei Li. Polymorphic PUF: Exploiting Reconfigurability of CPU+ FPGA SoC to Resist Modeling Attack. In *2017 Asian Hardware Oriented Security and Trust Symp. (AsianHOST)*, pages 43–48. IEEE, 2017.

[YHD$^+$16] 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 Computing Systems*, PP(99), 2016.

[YVDM14] Meng-Day Mandel Yu, Ingrid Verbauwhede, Srinivas Devadas, and David MRaihi. A Noise Bifurcation Architecture for Linear Additive Physical Functions. *Hardware-Oriented Security and Trust (HOST), 2014 IEEE Intl. Symp. on*, pages 124–129, 2014.