Strong Machine Learning Attack Against PUFs with No Mathematical Model

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

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

Abstract. Although numerous attacks revealed the vulnerability of different PUF families to non-invasive Machine Learning (ML) attacks, the question is still open whether all PUFs might be learnable. Until now, virtually all ML attacks rely on the assumption that a mathematical model of the PUF functionality is known a priori. However, this is not always the case, and attention should be paid to this important aspect of ML attacks. This paper aims to address this issue by providing a provable framework for ML attacks against a PUF family, whose underlying mathematical model is unknown. We prove that this PUF family is inherently vulnerable to our novel PAC (Probably Approximately Correct) learning framework. We apply our ML algorithm on the Bistable Ring PUF (BR-PUF) family, which is one of the most interesting and prime examples of a PUF with an unknown mathematical model. We practically evaluate our ML algorithm through extensive experiments on BR-PUFs implemented on Field-Programmable Gate Arrays (FPGA). In line with our theoretical findings, our experimental results strongly confirm the effectiveness and applicability of our attack. This is also interesting since our complex proof heavily relies on the spectral properties of Boolean functions, which are known to hold only asymptotically. Along with this proof, we further provide the theorem that *all* PUFs must have some challenge bit positions, which have larger influences on the responses than other challenge bits.

Keywords: Machine learning · PAC learning · Boosting technique · Fourier analysis · Physically Unclonable Functions (PUFs)

1 Introduction

Nowadays, it is broadly accepted that Integrated Circuits (ICs) are subject to overbuilding and piracy due to the adaption of authentication methods relying on insecure key storage techniques [24]. To overcome the problem of secure key storage, Physically Unclonable Functions (PUFs) have been introduced as promising solutions [15,30]. For PUFs, the manufacturing process variations lead eventually to instance-specific, and inherent physical properties that can generate virtually

B. Gierlichs and A.Y. Poschmann (Eds.): CHES 2016, LNCS 9813, pp. 391–411, 2016.
DOI: 10.1007/978-3-662-53140-2_19

unique *responses*, when the instance is given some *challenges*. Therefore, PUFs can be utilized as either device fingerprints for secure authentication or as a source of entropy in secure key generation scenarios. In this case, there is no need for permanent key storage, since the desired key is generated instantly upon powering up the device. Regarding the instance-specific, and inherent physical properties of the PUFs, they are assumed to be *unclonable* and *unpredictable*, and therefore trustworthy and robust against attacks [26]. However, after more than a decade of the invention of PUFs, the design of a really unclonable physical function is still a challenging task. Most of the security schemes relying on the notion of PUFs are designed based on a "design-break-patch" rule, instead of a thorough cryptographic approach.

Along with the construction of a wide variety of PUFs, several different types of attacks, ranging from non-invasive to semi-invasive attacks [18,19,33,39], have been launched on these primitives. Machine learning (ML) attacks are one of the most common types of non-invasive attacks against PUFs, whose popularity stems from their characteristics, namely being cost-effective and non-destructive. Moreover, these attacks require the adversary to *solely* observe the input-output (i.e., so called *challenge-response*) behavior of the targeted PUF. In this attack scenario, a relatively small subset of challenges along with their respective responses is collected by the adversary, attempting to come up with a model describing the challenge-response behavior of the PUF. In addition to heuristic learning techniques, e.g., what has been proposed in [33,34], the authors of [12–14] have proposed the probably approximately correct (PAC) learning framework to ensure the delivery of a model for prespecified levels of accuracy and confidence. One of the key results reported in [12–14] is that knowing about the mathematical model of the PUF functionality enables the adversary to establish a proper *hypothesis representation* (i.e., mathematical model of the PUF), and then try to PAC learn this representation. This gives rise to the question of whether a PUF can be PAC learned without prior knowledge of a precise mathematical model of the PUF.

Bistable Ring PUFs (BR-PUF) [7] and Twisted Bistable Ring PUFs (TBR-PUF) [37] are examples of PUFs, whose functionality cannot be easily translated to a precise mathematical model. In an attempt, the authors of [37,41] suggested simplified mathematical models for BR-PUFs and TBR-PUFs. However, their models do not precisely reflect the physical behavior of these architectures.

In this paper, we present a sound mathematical machine learning framework, which enables us to PAC learn the BR-PUF family (i.e., including BR- and TBR-PUFs) without knowing their precise mathematical model. Particularly, our framework contributes to the following novel aspects related to the security assessment of PUFs in general:

Exploring the inherent mathematical properties of PUFs. One of the most natural and commonly accepted mathematical representation of a PUF is a Boolean function. This representation enables us to investigate properties of PUFs, which are observed in practice, although they have not been precisely and mathematically described. One of these properties exhaustively studied in our paper is related to the "silent" assumption that each and every bit of a challenge has equal influence on the respective response of a PUF. We prove that this assumption is invalid for all PUFs. While this phenomenon has been already occasionally observed in practice and is most often attributed to implementation imperfections, we will give a rigorous mathematical proof on the existence of influential bit positions, which holds for every PUF.

Strong ML attacks against PUFs without available mathematical model. We prove that even in a worst case scenario, where the internal functionality of the BR-PUF family cannot be mathematically modeled, the challenge-response behavior of these PUFs can be PAC learned for given levels of accuracy and confidence.

Evaluation of the applicability of our framework in practice. In order to evaluate the effectiveness of our theoretical framework, we conduct extensive experiments on BR-PUFs and TBR-PUFs, implemented on a commonly used Field Programmable Gate Array (FPGA).

2 Notation and Preliminaries

This section serves as brief introduction into the required background knowledge and known results to understand the approaches taken in this paper. For some more complex topics we will occasionally refer the reader to important references.

2.1 PUFs

Note that elaborate and formal definitions as well as formalizations of PUFs are beyond the scope of this paper, and for more details on them we refer the reader to [3,4]. In general, PUFs are physical input to output mappings, which map given *challenges* to *responses*. Intrinsic properties of the physical primitive embodying the PUF determine the characteristics of this mapping. Two main classes of PUFs, namely strong PUFs and weak PUFs have been discussed in the literature [16]. In this paper we consider the strong PUFs, briefly called PUFs. Here we focus only on two characteristics of PUFs, namely unclonablity and unpredictability (i.e., so called unforgeability). Let a PUF be described by the mapping $f_{\mathrm{PUF}} : \mathcal{C} \to \mathcal{Y}$, where $f_{\mathrm{PUF}}(c) = y$. In this paper, we assume that the issue with noisy responses (i.e., the output is not stable for a given input) must have been resolved by the PUF manufacturer. For an *ideal* PUF, unclonablity means that for a given PUF f_{PUF} it is virtually impossible to create another physical mapping $g_{\mathrm{PUF}} \neq f_{\mathrm{PUF}}$, whose challenge-response behavior is *similar* to f_{PUF} [3].

Moreover, an ideal PUF is *unpredictable*. This property of PUFs is closely related to the notion of learnability. More precisely, given a single PUF f_{PUF} and a set of challenge response pairs (CRPs) $U = \{(c, y) \mid y = f_{\mathrm{PUF}}(c) \text{ and } c \in \mathcal{C}\}$, it is (almost) impossible to predict $y' = f_{\mathrm{PUF}}(c')$, where c' is a random challenge so that $(c', \cdot) \notin U$. In this paper we stick to this (simple, but) classical definition of unpredictability of a PUF, and refer the reader to [3,4] for more refined definitions.

2.2 Boolean Functions as representations of PUFs

Defining PUFs as mappings (see Sect. 2.1), the most natural mathematical model for them are Boolean functions over the finite field $\mathbb{F}_2$. Let $V_n = \{c_1, c_2, \ldots, c_n\}$ denote the set of Boolean attributes or variables, where each attribute can be ***true*** or ***false***, commonly denoted by "1" and "0", respectively. In addition, $C_n = \{0,1\}^n$ contains all binary strings with n bits. We associate each Boolean attribute c_i with two *literals*, i.e., c_i, and $\overline{c}_i$ (complement of c_i). An *assignment* is a mapping from V_n to $\{0,1\}$, i.e., the mapping from each Boolean attribute to either "0" or "1". In other words, an assignment is an n-bits string, where the i^{th} bit of this string indicates the value of c_i (i.e., "0" or "1").

An assignment is mapped by a Boolean formula into the set $\{0,1\}$. Thus, each Boolean attribute can also be thought of as a formula, i.e., c_i and $\overline{c}_i$ are two possible formulas. If by evaluating a Boolean formula under an assignment we obtain "1", it is called a *positive example* of the "concept represented by the formula" or otherwise a *negative example.* Each Boolean formula defines a respective Boolean function $f : C_n \rightarrow \{0,1\}$. The conjunction of Boolean attributes (i.e., a Boolean formula) is called a *term*, and it can be true or false ("1" or "0") depending on the value of its Boolean attributes. Similarly, a *clause* that is the disjunction of Boolean attributes can be defined. The number of literals forming a term or a clause is called its size. The size 0 is associated with only the term **true**, and the clause **false**.

In the related literature several representations of Boolean functions have been introduced, e.g., juntas, Monomials (M_n), Decision Trees (DTs), and Decision Lists (DLs), cf. [29,31].

A Boolean function depending on solely an unknown set of k variables is called a k-junta. A monomial $M_{n,k}$ defined over V_n is the conjunction of at most k clauses each having only one literal. A DT is a binary tree, whose internal nodes are labeled with a Boolean variable, and each leaf with either "1" or "0". A DT can be built from a Boolean function in this way: for each assignment a unique path form the root to a leaf should be defined. At each internal node, e.g., at the i^{th} level of the tree, depending on the value of the i^{th} literal, the labeled edge is chosen. The leaf is labeled with the value of the function, given the respective assignment as the input. The *depth* of a DT is the maximum length of the paths from the root to the leafs. The set of Boolean functions represented by decision trees of depth at most k is denoted by k-DT. A DL is a list L that contains r pairs $(f_1, v_1), \ldots, (f_r, v_r)$, where the Boolean formula f_i is a term and $v_i \in \{0,1\}$ with $1 \leq i \leq r-1$. For $i = r$, the formula f_r is the constant function $v_r = 1$. A Boolean function can be transformed into a decision list, where for a string $c \in C_n$ we have $L(c) = v_j$, where j is the smallest index in L so that $f_j(c) = 1$. k-DL denotes the set of all DLs, where each f_i is a term of maximum size k.

Linearity of Boolean Functions. Here, our focus is on *Boolean linearity*, which must not be confused with the linearity over other domains different from $\mathbb{F}_2$.

A linear Boolean function $f : \{0,1\}^n \to \{0,1\}$ features the following equivalent properties, cf. [29]:

- $\forall c, c' \in \{0,1\}^n : f(c + c') = f(c) + f(c')$
- $\exists a \in \{0,1\}^n : f(c) = a \cdot c$.

Equivalently, we can define a linear Boolean function f as follows. There is some set $S \subseteq \{1, \ldots, n\}$ such that $f(c) = f(c_1, c_2, \ldots, c_n) = \sum_{i \in S} c_i$.

Boolean linearity or linearity over $\mathbb{F}_2$ is closely related to the notion of correlation immunity. A Boolean function f is called ***k*-correlation immune**, if for any assignment c chosen randomly from $\{0,1\}^n$ it holds that $f(c)$ is independent of any k-tuple $(c_{i_1}, c_{i_1}, \ldots, c_{i_k})$, where $1 \le i_1 < i_2 < \cdots < i_k \le n$. Now let $\deg(f)$ denote the degree of the $\mathbb{F}_2$-polynomial representation of the Boolean function f. It is straightforward to show that such representation exists. Siegenthaler proved the following theorem, which states how correlation immunity can be related to the degree of f.

Theorem 1. (Siegenthaler Theorem [29,38]) *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, which is k-correlation immune, then* $\deg(f) \le n - k$.

Average Sensitivity of Boolean Functions. The Fourier expansion of Boolean functions serves as an excellent tool for analyzing them, cf. [29]. In order to define the Fourier expansion of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ we should first define an encoding scheme as follows. $\chi(0_{\mathbb{F}_2}) := +1$, and $\chi(1_{\mathbb{F}_2}) := -1$. 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

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

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. [29]. Next we define the **average sensitivity** of a Boolean function f as

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

2.3 Our Learning Model

The Probably Approximately Correct (PAC) model provides a firm basis for analyzing the efficiency and effectiveness of machine learning algorithms. We briefly introduce the model and refer the reader to [23] for more details. In the

PAC model the learner, i.e., the learning algorithm, is given a set of *examples* to generate with high probability an approximately correct hypothesis. This can be formally defined as follows. Let $F = \cup_{n \geq 1} F_n$ denote a *target concept class* that is a collection of Boolean functions defined over the *instance space* $C_n = \{0,1\}^n$. Moreover, according to an arbitrary probability distribution D on the instance space C_n each example is drawn. Assume that hypothesis $h \in F_n$ is a Boolean function over C_n, it is called an ε-approximator for $f \in F_n$, if

$$\Pr_{c \in_D C_n} [f(c) = h(c)] \geq 1 - \varepsilon.$$

Let the mapping $size : \{0,1\}^n \rightarrow \mathbb{N}$ associate a natural number $size(f)$ with a target concept $f \in F$ that is a measure of complexity of f under a target representation, e.g., k-DT. The learner is a polynomial-time algorithm denoted by $\mathtt{A}$, which is given labeled examples $(c, f(c))$, where $c \in C_n$ and $f \in F_n$. The examples are drawn independently according to distribution D. Now we can define strong and weak PAC learning algorithms.

Definition 1. *An algorithm* $\mathtt{A}$ *is called a* **strong** *PAC learning algorithm for the target concept class* F*, if for any* $n \geq 1$*, any distribution* D*, any* $0 < \varepsilon, \delta < 1$*, and any* $f \in F_n$ *the follwing holds. When* $\mathtt{A}$ *is given a polynomial number of labeled examples, it runs in time polynomial in* n*,* $1/\varepsilon$*,* $size(f)$*,* $1/\delta$*, and returns an* ε*-approximator for* f *under* D*, with probability at least* $1 - \delta$*.*

The weak learning framework was developed to answer the question whether a PAC learning algorithm with constant but insufficiently low levels of ε and δ can be useful at all. This notion is defined as follows.

Definition 2. *For some constant* $\delta > 0$ *let algorithm* $\mathtt{A}$ *return with probability at least* $1 - \delta$ *an* $(1/2 - \gamma)$*-approximator for* f*, where* $\gamma > 0$*.* $\mathtt{A}$ *is called a* **weak** *PAC learning algorithm, if* $\gamma = \Omega\left(1/p(n, size(f))\right)$ *for some polynomial* $p(\cdot)$*.*

The equivalence of weak PAC learning and strong PAC learning has been proved by Freund and Schapire in the early nineties in their seminal papers [9,35]. For that purpose *boosting* algorithms have been introduced.

Definition 3. *An algorithm* $\mathtt{B}$ *is called a boosting algorithm if the following holds. Given any* $f \in F_n$*, any distribution* D*,* $0 < \varepsilon, \delta < 1$*,* $0 < \gamma \leq 1/2$*, a polynomial number of labeled examples, and a weak learning algorithm* $\mathtt{WL}$ *returning an* $(1/2 - \gamma)$*-approximator for* f*, then* $\mathtt{B}$ *runs in time, which is polynomial in* n*,* $size(f)$*,* $1/\gamma$*,* $1/\varepsilon$*,* $1/\delta$ *and generates with probability at least* $1 - \delta$ *an* ε*-approximator for* f *under* D*.*

The construction of virtually all existing boosting algorithms is based primarily on the fact that if $\mathtt{WL}$ is given examples drawn from any distribution D', $\mathtt{WL}$ returns a $(1/2 - \gamma)$-approximator for f under D'. At a high-level, the skeleton of all such boosting algorithms is shown in Algorithm 1.

Algorithm 1. Canonical Booster

```
Require: Weak PAC learner WL, $0 < \varepsilon, \delta < 1$, $0 < \gamma \leq 1/2$, polynomial number of
    examples, $i$ that is the number of iterations
Ensure: Hypothesis $h$ that is an $\varepsilon$-approximator for $f$

$D_0 = D$, use WL to generate an approximator $h_0$ for $f$ under $D_0$
$k = 1$
while $k \leq i - 1$ do
    Build a distribution $D_k$ consisting of examples, where the previous approximators
        $h_0, \cdots, h_{k-1}$ can predict the value of $f$ poorly
    use WL to generate an approximator $h_k$ for $f$ under $D_k$
    $k = k + 1$
od
Combine the hypotheses $h_0, \cdots, h_{i-1}$ to obtain $h$, where each $h_i$ is an $(1/2 - \gamma)$-
    approximator for $f$ under $D_i$, and finally $h$ is an $\varepsilon$-approximator for $f$ under
    $D$
return $h$
```

2.4 Non-linearity of PUFs Over $\mathbb{F}_2$ and the Existence of Influential Bits

Section 2.2 introduced the notion of Boolean linearity. Focusing on this notion and taking into account the definition of PUFs mentioned in Sect. 2.1, now we prove the following theorem that is our first important result. For all PUFs, when represented as a Boolean function, it holds that their degree as $\mathbb{F}_2$-polynomial is strictly greater than one. This will then lead us to the following dramatic consequence. *There exists no PUF, in which all of its challenge bits have an equal influence.*

Theorem 2. *For every PUF* $f_{PUF} : \{0,1\}^n \rightarrow \{0,1\}$, *we have* $\deg(f_{PUF}) \geq 2$. *Consequently, for every PUF it holds that not all bit positions within respective challenges are equally influential in generating the corresponding response.*

Proof: Towards contradiction assume that f_{PUF} is Boolean linear over $\mathbb{F}_2$ and unpredictable. From the unpredictability of f_{PUF} it follows that the adversary has access to a set of CRPs $U = \{(c, y) \mid y = f_{\text{PUF}}(c) \text{ and } c \in \mathcal{C}\}$, which are chosen uniformly at random, however, the adversary has only a negligible probability of success to predict a new random challenge $(c', \cdot) \notin U$ (as he cannot apply f_{PUF} to this unseen challenge). Note that the size of U is actually polynomial in n. Now, by the definition of linearity over $\mathbb{F}_2$, cf. Sect. 2.2, we deduce that the only linear functions over $\mathbb{F}_2$ are the Parity functions, see also [29,38]. However, there are well-known algorithms to PAC learn Parity functions in general [8,20]. Thus, now we simply feed the right number of samples from our CRP set U into such a PAC learner. For the right parameter setting, the respective PAC algorithm delivers then with high probability an ε-approximator h for our PUF f_{PUF} such that $\Pr[f(c') = h(c')] \geq 1 - \varepsilon$. This means that with high probability, the response to every randomly chosen challenge can be calculated in polynomial time. This is of course a contradiction to the definition of f_{PUF}, being a PUF. Hence, f_{PUF} cannot be linear over $\mathbb{F}_2$. In other words, for every PUF f_{PUF} we have $\deg(f_{\mathrm{PUF}}) \geq 2$. Moreover, in conjunction with the above mentioned Siegenthaler Theorem, we deduce that every PUF is at most an $n-2$-correlation immune function, which indeed means that not all of its challenge bits have an equal influence on the respective PUF response. ■

Theorem 2 states that every PUF has some challenge bits, which have some larger influence on the responses than other challenge bits. We call these bits "loosely" as *influential bits*[1].

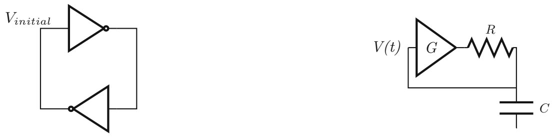
3 PUF Architectures

In this section, we explain the architectures of two intrinsic silicon PUFs, namely the BR- and TBR-PUFs, whose internal mathematical models are more complicated than other intrinsic PUF constructions. In an attempt, we apply simple models to describe the functionality of these PUFs. However, we believe that these models cannot completely reflect the real characteristics of the BR-PUF family, and their concrete, yet unknown model should be much more complex.

3.1 Memory-Based PUFs

BR-PUFs can be thought of as a combination of memory-based and delay-based PUFs. Memory-based PUFs exploit the settling state of digital memory circuits, e.g., SRAM cells [16,21] consisting of two inverters in a loop (see Fig. 1a) and two transistors for read and write operation. Due to manufacturing process variations the inverters have different electrical gains, when the cell is in the metastable condition. In the metastable condition the voltage of one of the inverters is equal to V_m, where V_m is an invalid logic level. Moreover, the invertes have different propagation delays due to the differences in their output resistance and load capacitance. One can model the SRAM cell architecture as a linear amplifier with gain G, when $V_{initial}$ is close to the metastable voltage V_m [40], see Fig. 1b. In order to predict the metastable behavior, we have [40]

$$V_{initial}(0) = V_m + V(0),$$



(a) (b)

Fig. 1. (a) The logical circuit of an SRAM cell. (b) The small signal model of bistable element in metastability

[1] Note that the existence of such influential bits has been also noticed by several other experimental research papers. However, none of them has been able to correctly and precisely pinpoint the mathematical origin of this phenomenon.

where $V(0)$ is a small signal offset from the metastable point. To derive $V(t)$ we can write the equation of the circuit as follows.

$$\frac{G \cdot V(t) - V(t)}{R} = C \cdot \frac{dV(t)}{dt}.$$

By solving this equation, we obtain $V(t) = V(0) \cdot e^{t/\tau_s}$, where $\tau_s = RC/G - 1$, c.f. [40]. The time required to reach a stable condition increases as $V_{initial}$ approaches the metastable point and $V(0)$ approaches 0. On the other hand, it can approach infinity, if $V(0) = 0$, however, in practice this is not the case due to the presence of noise. Nevertheless, there is no upper bound on the settling time of the SRAM cell to one of the stable states. Therefore, the settling state of the SRAM cells cannot be predicted after power-on. One can thus use the logical addresses of SRAM cells as different challenges and the state of the SRAM cells after power-on as PUF responses.

3.2 Bistable Ring PUF

SRAM PUFs are believed to be secure against modeling attacks. This can be explained by the fact that knowing the state of one SRAM PUF after power-on does not help the attacker to predict the response of other SRAM cells. However, in contrast to delay-based PUFs, e.g., arbiter PUFs [25], the challenge space of an SRAM PUF is not exponential. Therefore, if an adversary gets access to the initial values stored in the SRAM cells, the challenge-response behavior of the SRAM PUF can be emulated. In order to combine the advantages offered by delay-based PUFs and memory-based PUFs, namely, exponential challenge space and the unpredictability, a new architecture called BR-PUF was introduced by [7]. A BR-PUF consists of n stages (n is an even number), where each stage consists of two NOR gates, one demultiplexer and one multiplexer, see Fig. 2. Based on the value of the i^{th} bit of a

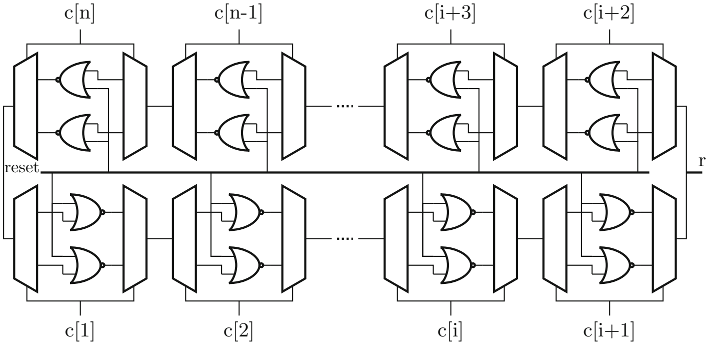


Fig. 2. The schematic of a BR-PUF with n stages. The response of the PUF can be read between two arbitrary stages. For a given challenge, the reset signal can be set low to activate the PUF. After a transient period, the BR-PUF might be settled to an allowed logical state.

challenge applied to the i^{th} stage, one of the NOR gates is selected. Setting the reset signal to low, the signal propagates in the ring, which behaves like an SRAM cell with a larger number inverters. The response of the PUF is a binary value, which can be read from a predefined location on the ring between two stages, see Fig. 2.

The final state of the inverter ring is a function of the gains and the propagation delays of the gates. According to the model of the SRAM circuit in the metastable state provided in Sect. 3.1, one might be able to extend the electrical model and analyze the behavior of the inverter ring. Applying a challenge, the ring may settle at a stable state after a oscillation time period. However, for a specific set of challenges the ring might stay in the metastable state for an infinite time, and the oscillation can be observed in the output of the PUF.

The analytical models of the metastable circuits introduced in Sect. 3.1 are valid for an ASIC implementation and respective simulations. Although few simulation results of BR-PUF are available in the literature, to the best of our knowledge there are no results for a BR-PUF implemented on an ASIC, and experimental results have been limited to FPGA implementations. In this case, the BR-PUF model can be further simplified by considering the internal architecture of the FPGAs. The NOR gates of the BR-PUF are realized by dedicated Lookup Tables (LUTs) inside an FPGA. The output of the LUTs are read from one of the memory cells of the LUT, which have always stable conditions. Hence, it can be assumed that there is almost no difference in the gains of different LUTs. As a result, the random behavior of the BR-PUF could be defined by the delay differences between the LUTs.

3.3 Twisted Bistable Ring PUF

Although the mathematical model of the functionality of a BR-PUF is unknown, it has been observed that this construction is vulnerable to bias and simple linear approximations [37]. Hence, the TBR-PUF, as an enhancement to BR-PUFs, has been introduced [37]. Similar to BR-PUFs, a TBR-PUF consists of n stages (n is an even number), where each stage consists of two NOR gates. In contrast to BR-PUF, where for a given challenge only one of the NOR gates in each stage is selected, all $2n$ gates are selected in a TBR-PUF. This can be achieved by placing two multiplexers before and two multiplexers after each stage and having feedback lines between different stages, see Fig. 3. As all NOR gates are always in the circuit, the challenge specific bias can be reduced.

4 PAC Learning of PUFs Without Prior Knowledge of Their Mathematical Model

When discussing the PAC learnability of PUFs as a target concept, two scenarios should be distinguished. First, the precise mathematical model of the PUF functionality is known, and hence, a hypothesis representation is known to learn the PUF. This scenario has been considered in several studies, e.g., [12–14], where different hypothesis representations have been presented for each individual PUF

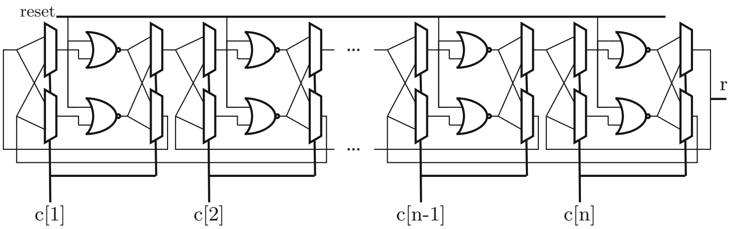


Fig. 3. The schematic of a TBR-PUF with n stages. The response of the PUF is read after the last stage. For a given challenge, the reset signal can be set low to activate the PUF. After a transient period, the BR-PUF might be settled to an allowed logical state.

family. Second, due to the lack of a precise mathematical model of the respective PUF functionality, to learn the PUF a more sophisticated approach is required. Therefore, the following question arises: is it possible to PAC learn a PUF family, even if we have no mathematical model of the physical functionality of the respective PUF family? We answer this question at least for the BR-PUF family. Our roadmap for answering this question, more specifically, the steps taken to prove the PAC learnability of BR-PUF family in the second scenario, is illustrated in Fig. 4. While theoretical insights into the notions related to the first two blocks have been presented in Sect. 2.4, which are valid for all PUF families, Sect. 4.1 provides more specific results for the BR-PUF family (i.e., According to these new insights, in Sect. 4.2 we eventually prove that BR-PUF family (which lack a precise mathematical model) can nevertheless be PAC learned (see last two blocks in Fig. 4).

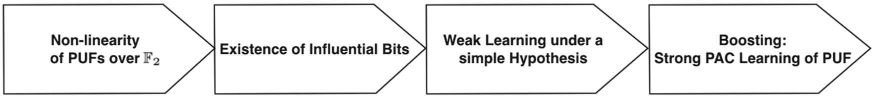

Non-linearity of PUFs over $\mathbb{F}_2$
Existence of Influential Bits
Weak Learning under a simple Hypothesis
Boosting: Strong PAC Learning of PUF

Fig. 4. Our roadmap for proving the PAC learnability of BR-PUF family, whose mathematical model is unknown

4.1 A Constant Upper Bound on the Number of Influential Bits

First, we reflect the fact that our Theorem 2 is in line with the empirical results obtained by applying heuristic approaches, which are reported in [37,42]. Although here we compare their results for BR- and TBR-PUFs with our results, our proof of having influential bits in PUF families in general, speaks for itself, and is one of the novel aspects of this paper.

In an attempt to assess the security of BR-PUFs, Yamamoto et al. have implemented BR-PUFs on several FPGAs to analyze the influence of challenge bits on the respective responses [42]. They have explicitly underlined the existence of influential bits, and found so called prediction rules. Table 1 summarizes their results, where for each type of the rules (monomials of different sizes) we report only the one with the highest estimated response prediction probability. In addition to providing evidence for the existence of influential bits, the size of the respective monomials is of particular importance for us. As shown in Table 1, their size is surprisingly small, i.e., only five.

Table 1. Statistical analysis of the 2048 CRPs, given to a 64-bit BR-PUF [42]. The first column shows the rule found in the samples, whereas the second column indicates the estimated probability of predicting the response.

Rule	Est. Pr.
$(c_1 = 0) \rightarrow y = 1$	0.684
$(c_9 = 0) \wedge (c_6 = 1) \rightarrow y = 1$	0.762
$(c_{25} = 0) \wedge (c_{18} = 1) \wedge (c_1 = 0) \rightarrow y = 1$	0.852
$(c_{27} = 0) \wedge (c_{25} = 0) \wedge (c_{18} = 1) \wedge (c_6 = 1) \rightarrow y = 1$	0.932
$(c_{53} = 0) \wedge (c_{51} = 0) \wedge (c_{45} = 0) \wedge (c_{18} = 1) \wedge (c_7 = 0) \rightarrow y = 1$	1

Similarly, the authors of [37] translate the influence of the challenge bits to the weights needed in artificial neural networks that represent the challenge-response behavior of BR-PUFs and the TBR-PUFs. They observed that there is a pattern in these weights, which models the influence of the challenge bits. It clearly reflects the fact that there are influential bits determining the response of the respective PUF to a given challenge. From the results presented in [37], we conclude that there is at least one influential bit, however, the precise number of influential bits has not been further investigated by the authors.

Inspired by the above results from [37,42], we conduct further experiments. We collect 30000 CRPs from BR-PUFs and TBR-PUFs implemented on Altera Cyclone IV FPGAs. In all of our PUF instances at least one influential bit is found, and the maximum number of influential bits (corresponding to the size of the monomials) is just a constant value in all cases. For the sake of readability, we present here only the results obtained for one arbitrary PUF instance.

Our results shown in Table 2 are not only aligned with the results reported in [37,42], but also reflect our previous theoretical findings. We could conclude this section as follows. There is at least one influential bit determining the response of a BR-PUF (respectively, TBR-PUF) to a given challenge. However, for the purpose of our framework their existence is not enough, and we need an upper bound on the number of influential bits.

Looking more carefully into the three different datasets, namely our own and the data reported in [37,42], we observe that the total number of influential bits is always only a very small value. Motivated by this commonly observed phenomenon, we compute for our PUFs (implemented on FPGAs) the average

Table 2. Our statistical analysis of the 30000 CRPs, given to a 64-bit BR-PUF. The first column shows the rule found in the sample, whereas the second column indicates the estimated probability of predicting the response.

Rule	Est. Pr.
$(c_{61} = 1) \rightarrow y = 1$	0.71
$(c_{11} = 1) \rightarrow y = 1$	0.72
$(c_{29} = 1) \rightarrow y = 1$	0.725
$(c_{39} = 1) \rightarrow y = 1$	0.736
$(c_{23} = 1) \rightarrow y = 1$	0.74
$(c_{46} = 1) \rightarrow y = 1$	0.745
$(c_{50} = 1) \rightarrow y = 1$	0.75
$(c_{61} = 1) \wedge (c_{23} = 1) \rightarrow y = 1$	0.82
$(c_{61} = 1) \wedge (c_{11} = 0) \rightarrow y = 1$	0.80
$(c_{23} = 1) \wedge (c_{46} = 1) \rightarrow y = 1$	0.86
$(c_{39} = 1) \wedge (c_{50} = 1) \rightarrow y = 1$	0.85
$(c_{61} = 1) \wedge (c_{11} = 1) \wedge (c_{29} = 1) \rightarrow y = 1$	0.88
$(c_{50} = 1) \wedge (c_{23} = 1) \wedge (c_{46} = 1) \rightarrow y = 1$	0.93
$(c_{50} = 1) \wedge (c_{23} = 1) \wedge (c_{46} = 1) \wedge (c_{39} = 0) \rightarrow y = 1$	0.97
$(c_{50} = 1) \wedge (c_{23} = 1) \wedge (c_{11} = 0) \wedge (c_{39} = 0) \wedge (c_{29} = 1) \rightarrow y = 1$	0.98
$(c_{50} = 1) \wedge (c_{23} = 1) \wedge (c_{46} = 1) \wedge (c_{39} = 0) \wedge (c_{29} = 1) \rightarrow y = 1$	0.99
$(c_{50} = 1) \wedge (c_{23} = 1) \wedge (c_{46} = 1) \wedge (c_{39} = 0) \wedge (c_{29} = 1) \wedge (c_{11} = 0) \rightarrow y = 1$	0.994
$(c_{50} = 1) \wedge (c_{23} = 1) \wedge (c_{46} = 1) \wedge (c_{39} = 0) \wedge (c_{29} = 1) \wedge (c_{61} = 0) \rightarrow y = 1$	0.995
$(c_{50} = 1) \wedge (c_{23} = 1) \wedge (c_{46} = 1) \wedge (c_{39} = 0) \wedge (c_{29} = 1) \wedge (c_{61} = 1) \wedge (c_{11} = 0) \rightarrow y = 1$	1

sensitivity of their respective Boolean functions[2]. Averaging over many instances of our BR-PUFs, we obtain the results shown in Table 3 (TBR-PUFs scored similarly). This striking result[3] lead us to the following plausible heuristic.

"Constant Average Sensitivity of BR-PUF family": *for all practical values of n it holds that the average sensitivity of a Boolean function associated with a physical n-bit PUF from the BR-PUF family is only a constant value.*

[2] As explained in Sect. 2.2, for a Boolean function f, the influence of a variable and the total average sensitivity can be calculated by employing Fourier analysis. However, in practice this analysis is computationally expensive. Instead, it suffices to simply approximate the respective average sensitivity. This idea has been extensively studied in the learning theory-related and property testing-related literature (see [22], for a survey). Here we describe how the average sensitivity of a Boolean function, representing a PUF, can be approximated. We follow the simple and effective algorithm as explained in [32]. The central idea behind their algorithm is to collect enough random pairs of labeled examples from the Boolean function, which have the following property: $(c, f(c))$ and $(c^{\oplus i}, f(c^{\oplus i}))$, i.e., the inputs differ on a single Boolean variable.

[3] Note that it is a known result and being folklore, cf. [29], that randomly chosen n-bit Boolean functions have an expected average sensitivity of exactly $n/2$.

Table 3. The average sensitivity of n-bit BR-PUFs.

n	The average sensitivity
4	1.25
8	1.86
16	2.64
32	3.6
64	5.17

Finally, some relation between the average sensitivity and the strict avalanche criterion (SAC) can be recognized, although we believe that the average sensitivity is a more direct metric to evaluate the security of PUFs under ML attacks.

4.2 Weak Learning and Boosting of BR-PUFs

The key idea behind our learning framework is the provable existence of influential bits for any PUF and the constant average sensitivity of BR-PUFs in our scenario. These facts are taken into account to prove the existence of weak learners for the BR-PUF family. We start with the following theorem (Theorem 3) proved by Friedgut [11].

Theorem 3. *Every Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ with $\mathrm{I}(f) = k$ can be ε-approximated by another Boolean function h depending on only a constant number of Boolean variables K, where $K = \exp\left((2 + \sqrt{2\varepsilon \log_2(4k/\varepsilon)/k})\frac{k}{\varepsilon}\right)$, and $\varepsilon > 0$ is an arbitrary constant.*

We explain now how Theorem 3 in conjunction with the results presented in Sect. 4.1 help us to prove the existence of a weak learner (Definition 2) for the BR-PUF family.

Theorem 4. *Every PUF from the BR-PUF family is weakly learnable.*

Proof: For an arbitrary PUF from the BR-PUF family, consider its associated but unknown Boolean function that is denoted by f_{PUF} (i.e., our target concept). Our weak learning framework has two main steps. In the first step, we identify a (weak) approximator for f_{PUF}, and in the second step this approximator is PAC learned (in a strong sense). Still, we can guarantee only that the total error of the learner does not exceed $1/2 - \gamma$, where $\gamma > 0$, as we start with a weak approximator of f_{PUF}. The first step relies on the fact that Theorem 2 ensures the existence of influential bits for f_{PUF}, while we can also upper bound $\mathrm{I}(f_{\mathrm{PUF}})$ by some small constant value k due to the Constant Average Sensitivity heuristic. According to the Theorem 3 there is a Boolean function h that is an ε-approximator of f_{PUF}, which depends only on a constant number of Boolean variables K since k and ε are constant values, independent of n. However, note that h depends on an unknown set of K variables. Thus, our Boolean function

h is a so called K-junta function, cf. [29]. More importantly, for constant K it is known that the K-junta function can be PAC learned by a trivial algorithm within $O\left(n^K\right)$ steps, cf. [2,5,6]. This PAC algorithm is indeed our algorithm WL that weakly learns f_{PUF}. Carefully choosing the parameters related to our approximators as well as the PAC learning algorithm, we ensure that WL returns a $1/2 - \gamma$-approximator for f_{PUF} and some $\gamma > 0$. ■

Applying now the canonical booster introduced in Sect. 2.3 to our WL proposed in the proof of Theorem 4 and according to Definition 3, our weak learning algorithm can be transformed into an efficient and strong PAC learning algorithm.

Corollary 1. *BR-PUFs are strong PAC learnable, regardless of any mathematical model representing their challenge-response behavior.*

5 Results

5.1 PUF Implementation

We implement BR and TBR-PUFs with 64 stages on an Altera Cyclone IV FPGA, manufactured on a 60nm technology [1]. It turns out that most PUF implementations are highly biased towards one of the responses. Therefore, we apply different manual routing and placement configurations to identify PUFs with a minimum bias in their responses. However, it is known that by reducing the bias in PUF responses, the number of noisy responses increases [27].

Finding and resolving the noisy responses are two of the main challenges in the CRP measurement process. In almost all PUF constructions it can be predicted, at which point in time a generated response is valid and can be measured. For instance, for an arbiter PUF one can estimate the maximum propagation delay (evaluation period) between the enable point and the arbiter. After this time period the response is in a valid logical level (either "0" or "1") and does not change, and afterwards by doing majority voting on the responses generated for a given challenge the stable CRPs can be collected. However, in the case of BR-PUF family, for a given challenge the settling time of the response to a valid logical level is not known a priori, see Fig. 5. Furthermore, it is not known whether the response to a given challenge would

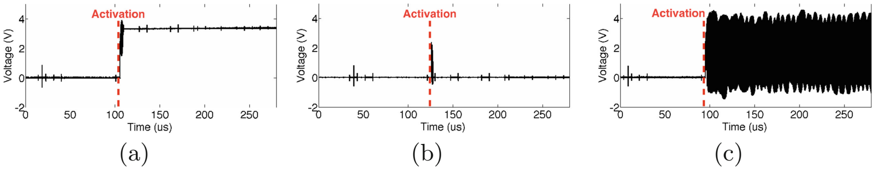
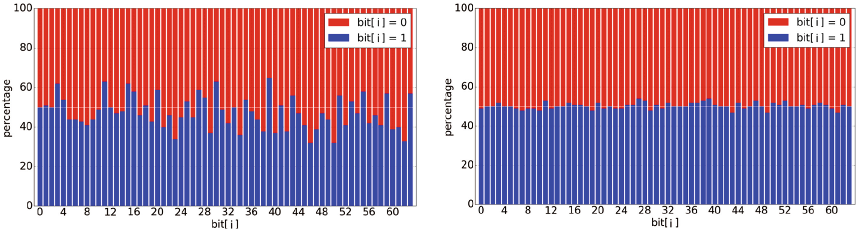


Fig. 5. The settling time of the BR-PUF response: (a) the PUF response after a transient time reaches a stable logical state "1". (b) after a transient time the PUF response is "0". (c) the PUF response does not settle and oscillates for an undefined time period.

not be unstable after observing the stable response during some time period (see Sect. 3.1). Therefore, the majority voting technique cannot be employed for BR-PUFs and TBR-PUFs. To deal with this problem, for a given challenge we read the response of the PUF at different points in time, where at each point in time 11 measurements are conducted additionally. We consider a response being stable, if it is the same at all these different measurement time points. Otherwise, the response is considered being unstable, and the respective CRP is excluded from our dataset.

In order to observe the impact of the existing influential bits on our PUF responses, first we apply a large set of challenges chosen uniformly at random, and then measure their respective responses. Afterwards, for both possible responses of the PUF (i.e., "0" and "1") we count the number of challenge bits, which are set to either "0" or "1", see Fig. 6. It can be seen that some challenge bits are more influential towards a certain response. These results are the basis for our statistical analysis presented in Sect. 4.1. We also repeat this experiment in the scenario, where the response of the PUF is unstable — in this case we observe almost no influential challenge bits. The most important conclusion that we can draw from these experiments is that a PUF with stable responses has at least one influential bit, which can already predict with low probability the response of the PUF to a respective challenge.



(a)

(b)

Fig. 6. The impact of the influential bits on the responses of the PUF: (a) the response of the PUF is "0". (b) unstable responses. Here the y-axis shows the percentage of the challenges, whose bits are set to either "0" or "1", whereas the x-axis shows the bit position.

5.2 ML Results

To evaluate the effectiveness of our learning framework, we conduct experiments on CRPs collected from our PUF, whose implementation is described in Sect. 5.1. As discussed and proved in Sect. 4, having influential bits enables us to define a prediction rule, where this rule can serve as a hypothesis representation, which fulfills the requirements of a weak learner. The algorithm WL proposed in the proof of the Theorem 4 relies on the PAC learnability of K-juntas, where K is a small constant. However, it is known that every efficient algorithm for learning K-DTs (i.e., the number of leaves is 2^K) is an efficient algorithm for learning K-juntas, see, e.g., [28]. Furthermore, it is known that DLs generalize K-DTs [31]. Moreover, a monomial $M_{n,K}$ is a very simple type of a K-junta, where only the conjunction of the relevant variables is taken into account. Therefore, for our experiments we decide to let our weak learning algorithms deliver DLs, Monomials, and DTs.

To learn the challenge-response behavior of BR- and TBR-PUFs using these representations, we use the open source machine learning software Weka [17]. One may argue that more advanced tools might be available, but here we only aim to demonstrate that publicly accessible, and off-the-shelf software can be used to launch our proposed attacks. All experiments are conducted on a MacBook Pro with 2.6 GHz Intel Core i5 processor and 10GB of RAM. To boost the prediction accuracy of the model established by our weak learners, we apply the Adaptive Boosting (AdaBoost) algorithm [10]; nevertheless, any other boosting framework can be employed as well. For Adaboost, it is known that the error of the final model delivered by the boosted algorithm after T iteration is theoretically upper bounded by $\prod_{t=1}^{T} \sqrt{1-4\gamma^2}$, c.f. [36]. To provide a better understanding of the relation between K, the number of iterations, and the theoretical bound on the error of the final model, a corresponding graph[4] is shown in Fig. 7.

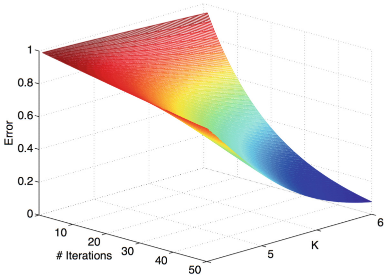


Fig. 7. The relation between the theoretical upper bound on the error of the final model returned by Adaboost, the number of iterations, and K. The graph is plotted for $k = 2$, $\varepsilon' = 0.01$, and $n = 64$. Here, $\varepsilon' = 0.01$ denotes the error of the K-junta learner.

Our experiments in Weka consist of a training phase and a testing phase. In the training phase a model is established from the training data based on the chosen representation. Afterwards, the established model is evaluated on the test set, which contains an unseen subset of CRPs. The size of the training sets in our

[4] Note that at first glance the graph may seem odd as after a few iterations the error is close to 1, although we start from a weak learner, whose error rate is strictly below 0.5. As explained in [36, pp. 57–60], and shown in their Fig. 3.1, this is due to Adaboosts's theoretical worst-case analysis, which is only asymptotically (in T) meaningful.

Table 4. Experimental results for learning 64-bit BR-PUF and TBR-PUF, when $m = 100$. The accuracy $(1 - \varepsilon)$ is reported for three weak learners. The first row shows the accuracy of the weak learner, whereas the other rows show the accuracy of the boosted learner.

# boosting iterations	BR-PUF			TBR-PUF		
	M_n	DT	DL	M_n	DT	DL
0 (no boosting)	54.48 %	66.79 %	67.24 %	65.18 %	72.29 %	74.84 %
10	67.12 %	74.25 %	76.99 %	76.96 %	79.22 %	81.36 %
20	77.53 %	80.53 %	80.89 %	82.05 %	85.73 %	86.71 %
30	81.32 %	83.13 %	83.14 %	84.93 %	88.34 %	89.4 %
40	82.65 %	83.91 %	84.6 %	88.11 %	89.67 %	90.22 %
50	82.65 %	85.62 %	85.5 %	90.05 %	89.69 %	91.58 %

Table 5. Experimental results for $m = 1000$ (the same setting as for the Table 4).

# boosting iterations	BR-PUF			TBR-PUF		
	M_n	DT	DL	M_n	DT	DL
0 (no boosting)	63.73 %	75.69 %	84.59 %	64.9 %	75.6 %	84.34 %
10	81.09 %	85.49 %	94.2 %	79.9 %	87.12 %	95.05 %
20	89.12 %	91.08 %	96.64 %	88.28 %	91.57 %	97.89 %
30	93.24 %	93.24 %	97.50 %	93.15 %	93.9 %	98.75 %
40	95.69 %	94.28 %	97.99 %	96.73 %	95.05 %	99.13 %
50	96.80 %	95.04 %	98.32 %	98.4 %	95.96 %	99.37 %

experiments are 100 and 1000, whereas the test set contains 30000 CRPs. Our experiments demonstrate that the weak learning of our test set always results in the delivery of a model with more than 50 % accuracy as shown in the first rows of Tables 4 and 5.

By boosting the respective models with AdaBoost, the accuracy is dramatically increased, see Tables 4 and 5. It can be observed that after 50 iterations of Adaboost applied to the weak model generated from 100 CRPs, the prediction accuracy of the boosted model is increased to more than 80 % for all three representations. By increasing the number of samples to 1000 CRPs, the prediction accuracy is further increased up to 98.32 % for learning the BR-PUFs, and 99.37 % for learning the TBR-PUFs under DL representations. It is interesting to observe that the simplest representation class, i.e., Monomials clearly present the greatest advantage given by the boosting technique. As explained in [36] this is due to avoiding any overfitting tendency.

6 Conclusion

As a central result, which speaks for itself, we have proved that in general the responses of all PUF families are not equally determined by each and every bit of their respective challenges. Moreover, the present paper has further addressed the issue of strong PAC learning of the challenge-response behavior of PUFs, whose functionality lacks a precise mathematical model. We have demonstrated that representing BR- and TBR-PUFs by Boolean functions, we are able to precisely describe the characteristics of these PUFs as observed in practice. This fact results in developing a new and generic machine learning framework that strongly PAC learns the challenge-response behavior of the BR-PUF family. The effectiveness and applicability of our framework have also been evaluated by conducting extensive experiments on BR-PUFs and TBR-PUFs implemented on FPGAs, similar to experimental platforms used in the most relevant literature.

Last but not least, although our strong PAC learning framework has its own novelty value, we feel that our Theorem 3 and the precise mathematical description of the characteristics of BR-PUFs and TBR-PUFs are the most important aspects of our paper. We strongly believe that this description can help to fill the gap between the mathematical design of cryptographic primitives and the design of PUFs in real world. As an evidence thereof, we feel that the Siegenthaler Theorem and the Fourier analysis that are well-known and widely used in modern cryptography may provide special insights into the physical design of secure PUFs in the future.

Acknowledgements. We would like to thank Prof. Dr. Frederik Armknecht for the fruitful discussion as well as pointing out the Siegenthaler's paper. Furthermore, the authors greatly appreciate the support that they received from Helmholtz Research School on Security Technologies.

References

1. Altera: Cyclone IV Device Handbook. Altera Corporation, San Jose (2014)
2. Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1988)
3. Armknecht, F., Maes, R., Sadeghi, A., Standaert, O.X., Wachsmann, C.: A formalization of the security features of physical functions. In: 2011 IEEE Symposium on Security and Privacy (SP), pp. 397–412 (2011)
4. Armknecht, F., Moriyama, D., Sadeghi, A.R., Yung, M.: Towards a unified security model for physically unclonable functions. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 271–287. Springer, Heidelberg (2016)
5. Arvind, V., Köbler, J., Lindner, W.: Parameterized learnability of k-juntas and related problems. In: Hutter, M., Servedio, R.A., Takimoto, E. (eds.) ALT 2007. LNCS (LNAI), vol. 4754, pp. 120–134. Springer, Heidelberg (2007)
6. Blum, A.L., Langley, P.: Selection of relevant features and examples in machine learning. Artif. Intell. **97**(1), 245–271 (1997)
7. Chen, Q., Csaba, G., Lugli, P., Schlichtmann, U., Rührmair, U.: The bistable ring PUF: a new architecture for strong physical unclonable functions. In: 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 134–141. IEEE (2011)
8. Fischer, P., Simon, H.U.: On learning ring-sum-expansions. SIAM J. Comput. **21**(1), 181–192 (1992)

9. Freund, Y.: Boosting a weak learning algorithm by majority. Inf. Comput. **121**(2), 256–285 (1995)
10. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comp. Syst. Sci. **55**(1), 119–139 (1997)
11. Friedgut, E.: Boolean functions with low average sensitivity depend on few coordinates. Combinatorica **18**(1), 27–35 (1998)
12. Ganji, F., Tajik, S., Seifert, J.P.: Let me prove it to you: RO PUFs are provably learnable. In: The 18th Annual International Conference on Information Security and Cryptology (2015)
13. Ganji, F., Tajik, S., Seifert, J.-P.: Why attackers win: on the learnability of XOR arbiter PUFs. In: Conti, M., Schunter, M., Askoxylakis, I. (eds.) TRUST 2015. LNCS, vol. 9229, pp. 22–39. Springer, Heidelberg (2015)
14. Ganji, F., Tajik, S., Seifert, J.P.: PAC learning of arbiter PUFs. J. Cryptographic Eng. Spec. Sect. Proofs **2014**, 1–10 (2016)
15. Gassend, B., Clarke, D., Van Dijk, M., Devadas, S.: Silicon physical random functions. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 148–160 (2002)
16. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
17. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD Explor. Newslett. **11**(1), 10–18 (2009)
18. Helfmeier, C., Boit, C., Nedospasov, D., Seifert, J.P.: Cloning physically unclonable functions. In: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 1–6 (2013)
19. Helfmeier, C., Nedospasov, D., Tarnovsky, C., Krissler, J.S., Boit, C., Seifert, J.P.: Breaking and entering through the silicon. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, pp. 733–744. ACM (2013)
20. Helmbold, D., Sloan, R., Warmuth, M.K.: Learning integer lattices. SIAM J. Comput. **21**(2), 240–266 (1992)
21. Holcomb, D.E., Burleson, W.P., Fu, K.: Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In: Proceedings of the Conference on RFID Security, vol. 7 (2007)
22. Kalai, G., Safra, S.: Threshold phenomena and influence: perspectives from mathematics, computer science, and economics. In: Computational Complexity and Statistical Physics, Santa Fe Institute Studies on the Sciences of Complexity, pp. 25–60 (2006)
23. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press, Cambridge (1994)
24. Koushanfar, F.: Hardware metering: a survey. In: Tehranipoor, M., Wang, C. (eds.) Introduction to Hardware Security and Trust, pp. 103–122. Springer, New York (2012)
25. Lee, J.W., Lim, D., Gassend, B., Suh, G.E., Van Dijk, M., Devadas, S.: A technique to build a secret key in integrated circuits for identification and authentication applications. In: 2004 Symposium on VLSI Circuits. Digest of Technical Papers, pp. 176–179 (2004)
26. Maes, R.: Physically Unclonable Functions: Constructions, Properties and Applications. Springer, Heidelberg (2013)

27. Maes, R., van der Leest, V., van der Sluis, E., Willems, F.: Secure key generation from biased PUFs. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 517–534. Springer, Heidelberg (2015)
28. Mossel, E., O'Donnell, R., Servedio, R.A.: Learning functions of k relevant variables. J. Comp. Syst. Sci. **69**(3), 421–434 (2004)
29. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press, Cambridge (2014)
30. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science **297**(5589), 2026–2030 (2002)
31. Rivest, R.L.: Learning decision lists. Mach. Learn. **2**(3), 229–246 (1987)
32. Ron, D., Rubinfeld, R., Safra, M., Samorodnitsky, A., Weinstein, O.: Approximating the influence of monotone boolean functions in $O(\sqrt{n})$ query complexity. ACM Trans. Comput. Theory (TOCT) **4**(4), 11 (2012)
33. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling attacks on physical unclonable functions. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 237–249 (2010)
34. Saha, I., Jeldi, R.R., Chakraborty, R.S.: Model building attacks on physically unclonable functions using genetic programming. In: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 41–44. IEEE (2013)
35. Schapire, R.E.: The strength of weak learnability. Mach. Learn. **5**(2), 197–227 (1990)
36. Schapire, R.E., Freund, Y.: Boosting: Foundations and Algorithms. MIT Press, Cambridge (2012)
37. Schuster, D., Hesselbarth, R.: Evaluation of bistable ring PUFs using single layer neural networks. In: Holz, T., Ioannidis, S. (eds.) Trust 2014. LNCS, vol. 8564, pp. 101–109. Springer, Heidelberg (2014)
38. Siegenthaler, T.: Correlation-immunity of nonlinear combining functions for cryptographic applications (Corresp.). IEEE Trans. Inf. Theory **30**(5), 776–780 (1984)
39. Tajik, S., Dietz, E., Frohmann, S., Seifert, J.-P., Nedospasov, D., Helfmeier, C., Boit, C., Dittrich, H.: Physical characterization of arbiter PUFs. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 493–509. Springer, Heidelberg (2014)
40. Weste, N.H.E., Harris, D.: CMOS VLSI Design: A Circuits and Systems Perspective, 4th edn. Addison Wesley, Boston (2010)
41. Xu, X., Rührmair, U., Holcomb, D.E., Burleson, W.P.: Security evaluation and enhancement of bistable ring PUFs. In: Mangard, S., Schaumont, P. (eds.) Radio Frequency Identification. LNCS, vol. 9440, pp. 3–16. Springer, Heidelberg (2015)
42. Yamamoto, D., Takenaka, M., Sakiyama, K., Torii, N.: Security evaluation of Bistable Ring PUFs on FPGAs using differential and linear analysis. In: 2014 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 911–918 (2014)