PUF Modeling Attacks on Simulated and Silicon Data

Ulrich Rührmair, Jan Sölter, Frank Sehnke, Xiaolin Xu, Ahmed Mahmoud, Vera Stoyanova, Gideon Dror, Jürgen Schmidhuber, Wayne Burleson, *Fellow, IEEE,* and Srinivas Devadas, *Fellow, IEEE.*

***Abstract*—We discuss numerical modeling attacks on several proposed Strong Physical Unclonable Functions (PUFs). Given a set of challenge-response pairs (CRPs) of a Strong PUF, the goal of our attacks is to construct a computer algorithm which behaves indistinguishably from the original PUF on almost all CRPs. If successful, this algorithm can subsequently impersonate the Strong PUF, and can be cloned and distributed arbitrarily. It breaks the security of any applications that rest on the Strong PUF's unpredictability and physical unclonability. Our method is less relevant for other PUF types such as Weak PUFs; see Section I-B for a detailed discussion of this topic.**

The Strong PUFs that we could attack successfully include standard Arbiter PUFs of essentially arbitrary sizes, and XOR Arbiter PUFs, Lightweight Secure PUFs, and Feed-Forward Arbiter PUFs up to certain sizes and complexities. We also investigate the hardness of certain Ring Oscillator PUF architectures in typical Strong PUF applications. Our attacks are based upon various machine learning techniques, including a specially tailored variant of Logistic Regression and Evolution Strategies.

Our results are mostly obtained on CRPs from numerical simulations that use established digital models of the respective PUFs. For a subset of the considered PUFs — namely standard Arbiter PUFs and XOR Arbiter PUFs — we also lead proofs of concept on silicon data from both FPGAs and ASICs. Over four million silicon CRPs are used in this process. The performance on silicon CRPs is very close to simulated CRPs, confirming a conjecture from earlier versions of this work. Our findings lead to new design requirements for secure electrical Strong PUFs, and will be useful to PUF designers and attackers alike.

***Index Terms*—Physical Unclonable Functions, Machine Learning, Cryptanalysis, Physical Cryptography**

I. Introduction

A. Motivation and Background

Electronic devices are now pervasive in our everyday life. This makes them an accessible target for adversaries, leading to a host of security and privacy issues. Classical cryptography offers several measures against these problems, but they all rest on the concept of a secret binary key: It is assumed that the devices can contain a piece of information that is, and remains, unknown to the adversary. Unfortunately, it can be difficult to uphold this requirement in practice. Physical attacks such as invasive, semi-invasive, or side-channel attacks, as well as software attacks like API-attacks and viruses, can lead to key exposure and full security breaks. The fact that the devices should be inexpensive, mobile, and cross-linked aggravates the problem.

The described situation was one motivation that led to the development of *Physical Unclonable Functions (PUFs)*. A PUF is a (partly) disordered physical system P that can be challenged with so-called external stimuli or challenges C_i, upon which it reacts with corresponding responses termed R_{C_i}. Contrary to standard digital systems, a PUF's responses shall depend on the nanoscale structural disorder present in the PUF. This disorder cannot be cloned or reproduced exactly, not even by its original manufacturer, and is unique to each PUF. As PUF responses can be noisy, suitable error correction techniques like fuzzy extractors [13] may be applied in practice to obtain stable outputs R'_{C_i}. Assuming successful error compensation, any PUF P can be regarded as an individual function F_P that maps challenges C_i to (stable) responses R'_{C_i} (compare [41]).

Due to its complex and disordered structure, a PUF can avoid some of the shortcomings associated with digital keys. For example, it is usually harder to read out, predict, or derive its responses than to obtain the values of digital keys stored in non-volatile memory. This fact has been exploited for various PUF-based security protocols. Prominent examples include schemes for identification and authentication [34], [15], key exchange or digital rights management purposes [16].

B. Modeling Attacks and Different PUF Types

There are several subtypes of PUFs, each with its own applications and security features. Three established types, which must explicitly be distinguished in this paper, are *Strong PUFs* [34], [15] [1] *Controlled PUFs* [16], and *Weak PUFs* [18], [20], also called *Physically Obfuscated Keys (POKs)* [14]. [2] For an exact differentiation, we refer the reader to

Ulrich Rührmair is with the Technische Universität München, Arcisstr. 21, 80333 München, Germany. E-mail: ruehrmair@in.tum.de

Jan Sölter is with the Technische Universität München, Germany, and the Freie Universität Berlin, Germany.

Frank Sehnke, Ahmed Mahmoud, Vera Stoyanova are with the Technische Universität München, Germany.

Jürgen Schmidhuber is with the Technische Universität München, Germany, and the University of Lugano, SUPSI, and IDSIA, all Switzerland.

Gideon Dror is with the Academic College of Tel-Aviv-Yaffo, Israel, and Yahoo research, Israel.

Xiaolin Xu and Wayne Burleson are with the University of Massachusetts Amherst, USA.

Srinivas Devadas is with the Massachusetts Institute of Technology, USA.

[1] Strong PUFs have sometimes also been referred to as Physical Random Functions [14].

[2] We would like to *stress* that the term "Weak PUF" and "Strong PUF" are not to be understood in any pejorative or judgemental sense. They are not meant to indicate that one PUF-type would be superior or inferior to another. We merely follow a terminology that had originally been introduced by Guajardo, Kumar, Schrijen and Tuyls [18], and which has later been developed further by Rührmair et al. in [42], [37], [41], [38].

earlier versions of this work [41], or a recent survey article by Rührmair, Devadas and Koushanfar [38]. We stress that the attacks presented in this paper do **not** apply to all of these three types in the same manner, as detailed below.

In general, modeling attacks on PUFs presume that an adversary Eve has, in one way or the other, collected a subset of all CRPs of the PUF. She then tries to derive a numerical model from this data, i.e., a computer algorithm which correctly predicts the PUF's responses to arbitrary challenges with high probability. Machine learning (ML) techniques are a natural and powerful tool for this task [14], [24], [31], [26], [45]. How the required CRPs can be collected, and how relevant our modeling attacks are in practice, very strongly depends on the considered type of PUF, however.

1) Strong PUFs: Strong PUFs are PUFs with very many possible challenges and a complex input-output relation [41], [38], [37]. They are the PUF class for which our modeling attacks have been designed originally, and to which they are best applicable. The reason is that Strong PUFs usually have no protection mechanisms that restrict Eve applying challenges or in reading out their responses [38], [37], [41]. Their responses are usually not post-processed on chip in a protected environment [34], [46], [27], [17], [23], [25]. Most electrical Strong PUFs further operate at frequencies of a few MHz [23]. Therefore even short physical access periods enable Eve to read-out and collect many CRPs. Another potential CRP source is simple protocol eavesdropping, for example on standard Strong PUF-based identification protocols, where the CRPs are sent in the clear [34]. Please note that both eavesdropping on responses as well as physical access to the PUF is part of the established, general attack model for PUFs.

Once a predictive model for a Strong PUF has been derived, the two main security features of a Strong PUF no longer hold: The PUF is no longer *unpredictable* for parties that are not in physical possession of the PUF; and the *physical unclonability* of the PUF is overcome by the fact that the digital simulation algorithm can be cloned and distributed arbitrarily. Any Strong PUF protocol which is built on these two features is then no longer secure. This includes any standard, widespread Strong PUF protocols known to the authors. [3]

For example, if Eve can use her intermediate physical access in a PUF-based key exchange protocol [11], [4] to derive a predictive model of the PUF, she can later predict the key that was exchanged between the honest parties. A similar effect occurs in 1-out-of-2 oblivious transfer (OT) protocols [36], [4]: If the OT-receiver can derive a numerical model of the PUF before he physically transfers the PUF to the OT-sender, he can later break the security of the sender, and learn *both* transfered bits b_0 and b_1. Also in the CRP-based, standard identification protocols for Strong PUFs [33], [34], a numerical model can be used to impersonate the original PUF.

Concerning applications where the form factor of the PUF may play a role, such as smartcards, we stress that the very simple additive simulation models derived in this paper can be implemented in similar environments as the original PUFs, and with a relatively small number of gates. An active fraudster can come so close to the original form factor in a newly set-up, malicious smartcard hardware that the difference is very difficult to notice in practice.

2) Controlled PUFs: Controlled PUFs are a second PUF-type, which consists of an underlying Strong PUF with a surrounding control logic [41], [38]. The challenge-response interface of the Strong PUF is not directly accessible, but is protected by the logic. Any challenges applied to the Controlled PUF are pre-processed by the logic before they are input to the Strong PUF, and any responses of the Strong PUF are postprocessed by the logic before they are being output by the Controlled PUF. Both the pre- and post-processing step can add significantly to the security of a Controlled PUF [16].

For any adversary that is restricted to non-invasive CRP measurement, Controlled PUFs successfully disable modeling attacks if the control logic uses a secure one-way hash over the outputs of the underlying Strong PUF. We note that this requires internal error correction of the Strong PUF outputs inside the Controlled PUF, since they are inherently noisy [16]. Furthermore, it introduces a new, additional presumption, namely the security of the applied one-way hash function.

Successful application of our techniques to a Controlled PUF only becomes possible if Eve can probe the internal, digital response signals of the underlying Strong PUF on their way to the control logic. Even though this is a significant assumption, probing digital signals is still easier than measuring continuous analog parameters within the underlying Strong PUF, for example determining its delay values. Note again that physical access to the PUF is part of the natural attack model on PUFs, as mentioned above. If a Controlled PUF has been modeled, the same effects for protocols resting on their unpredictability and physical unclonability apply that have been described in the last Section I-B1.

3) Weak PUFs: Weak PUFs (or POKs) are PUFs with few, fixed challenges, in the extreme case with just one challenge [41], [38]. It is usually assumed that their response(s) remain inside the PUF-carrying hardware, for example for the derivation of a secret key, and are not easily accessible for external parties. Weak PUFs are the PUF class that is the least susceptible to the presented modeling attacks.

We stress that our attacks only apply to them under relatively rare and special circumstances: namely if a Strong PUF, embedded in some hardware system and with a not publicly accessible CRP interface, is used to implement the Weak PUF. This method has been suggested in [14], [46]. Thereby only a few (of the very many possible) challenges of the Strong PUF are used for internal key derivation. Our attacks make sense in this context only in the special case that the Strong PUF challenges C_i^* that are used in the key derivation process are not yet fixed in the hardware at the time of fabrication, but are selected later on. For one reason

[3]One sole potential exception are a few recent bit commitment protocols for PUFs that were explicitly designed for the so-called *"bad PUF model"* or the *"malicious PUF model"*. They promise to uphold security even if one or all used PUFs are *not* unpredictable (see partly van Dijk and Rührmair [12] and mainly Damgard and Scafuro [8]). At least some of these protocols are relatively non-standard in a number of aspects, however, such as the assumed input/output lengths of the used PUFs. Asides from these two special protocols, all other practically relevant, widespread Strong PUF schemes straightforwardly break down if the main security feature of the Strong PUF is violated by a modeling attack, namely their unpredictability.

or another, the adversary may learn about these challenges at a point in time that lies after his point of physical access to the PUF. In this case, machine learning and modeling of the Strong PUF can help the adversary to derive the key, even though the points in time where he has access to the PUF and where he learns the challenges C_i^* strictly differ. In order to make our ML methods applicable in this case, one must assume that the adversary was able to collect many CRPs of the Strong PUF, for example by physically probing the internal digital response signals of the Strong PUF to randomly injected challenges, or by malware that abuses internal access to the underlying Strong PUF's interface. We comment that the latter scenarios obviously represent very strong attack models. Under comparable circumstances also many standard Weak PUFs and other secret key based architectures break down.

In any other cases than the above, our modeling attacks will not be relevant for Weak PUFs. This means that they are not applicable to the majority of current Weak PUF implementations, including the Coating PUF [47], SRAM PUF [18], Butterfly PUF [22], and similar architectures.

We conclude by the remark that this should not lead to the impression that Weak PUFs are necessarily more secure than other PUFs. Other attack strategies can be applied to them, including invasive, side-channel and virus attacks, but they are not the topic of this paper. For example, probing the output of the SRAM cell prior to storing the value in a register can break the security of the cryptographic protocol that uses these outputs as a key. Also physical cloning strategies for certain Weak PUFs have been reported recently [19]. Finally, we comment that attacking a Controlled PUF via collecting CRPs from the underlying Strong PUF requires substantially more signal probing than breaking a Weak PUF that possesses just one challenge.

C. Related Work

This article is an extended journal version of Rührmair et al. [41] from CCS'10. Early work on PUF modeling attacks, such as [17], [24], [31], [26], described successful attacks on standard Arbiter PUFs and on Feed-Forward Arbiter PUFs with one loop. But these approaches did not generalize to Feed-Forward Arbiter PUFs with more than two loops. The XOR Arbiter PUF, Lightweight PUF, Feed-Forward Arbiter PUF with more than two Feed-Forward Loops, and Ring Oscillator PUF had not been cryptanalyzed until the first version of this work [41]. Further, no scalability analyses of the required CRPs and computation times had been performed in any earlier works. In comparison to the first version of this article [41], the main novelty is that results on a very large database of silicon CRPs from ASICs and FPGAs have been added. The new result settles an open question from the first version of this work [41], showing that our findings on numerically simulated CRPs carry over with very little performance loss to the silicon case.

Since the publication of the earlier version of this article [41], some works have addressed the problem of the ML-susceptibility of Strong PUFs. For example, Majzoobi et al. [30] describe an approach to improve the resilience of PUFs against modeling attacks in identification protocols by certain hardware and protocol measures, assuming a Controlled PUF environment. Under the assumptions that the internal digital signals can be probed, however, similar attacks apply to this construction as described in this paper (compare Section I-B). Furthermore, Yu et al. [49] have described the use of PUFs for internal key derivation in the context of machine learning. Again, this paper is besides the focus of this work, which concentrates on Strong PUFs (compare Section I-B).

D. Organization of the Paper

The paper is organized as follows. We describe the methodology of our ML experiments in Section II. In Sections III to VII, we present our ML results for various Strong PUF candidates on simulated, noise-free CRP data. These sections deal with Arbiter PUFs, XOR Arbiter PUFs, Lightweight Arbiter PUFs, Feed-Forward Arbiter PUFs and Ring Oscillator PUFs, in sequence. Section VIII deals with the effect of randomly injected noise in the simulated CRP data. Section IX carries our a very detailed proof of concept for silicon data from FPGA and ASICs. We conclude with a summary and discussion of our results in Section X.

II. Methodology Section

A. Employed Machine Learning Methods

We evaluated various machine techniques prior to our experiments, including Support Vector Machines (SVMs), Logistic Regression (LR), Evolution Strategies (ES), and briefly also Neural Nets and Sequence Learning. The approaches in the following two sections performed best and are applied throughout the paper.

1) Logistic Regression: Logistic Regression (LR) is a well-investigated supervised machine learning framework, which has been described, for example, in [2]. In its application to PUFs with single-bit outputs, each challenge $C = b_1 \cdots b_k$ is assigned a probability $p\,(C, t \,|\, \vec{w})$ that it generates a output $t \in \{-1, 1\}$ (for technical reasons, one makes the convention that $t \in \{-1, 1\}$ instead of $\{0, 1\}$). The vector $\vec{w}$ thereby encodes the relevant internal parameters, for example the particular runtime delays, of the individual PUF. The probability is given by the logistic sigmoid acting on a function $f(\vec{w})$ parametrized by the vector $\vec{w}$ as $p\,(C, t \,|\, \vec{w}) = \sigma(tf) = (1 + e^{-tf})^{-1}$. Thereby f determines through $f = 0$ a decision boundary of equal output probabilities. For a given training set $\mathcal{M}$ of CRPs the boundary is positioned by choosing the parameter vector $\vec{w}$ in such a way that the likelihood of observing this set is maximal, respectively the negative log-likelihood is minimal:

$$\begin{aligned} \hat{\vec{w}} &= argmin_{\vec{w}}\ l(\mathcal{M}, \vec{w}) \\ &= argmin_{\vec{w}} \sum_{(C,\,t) \in\, \mathcal{M}} -ln\left(\sigma\left(tf(\vec{w}, C)\right)\right) \end{aligned} \qquad (1)$$

As there is no analytical solution to determine the optimal parameter vector $\hat{\vec{w}}$, it has to be optimized iteratively, e.g., using the gradient information

$$\nabla l(\mathcal{M}, \vec{w}) = \sum_{(C,\,t) \in\, \mathcal{M}} t(\sigma(tf(\vec{w}, C)) - 1)\nabla f(\vec{w}, C) \qquad (2)$$

From the different optimization methods which we tested in our ML experiments (standard gradient descent, iterative reweighted least squares, RProp [2] [35]), RProp gradient descent performed best. Logistic regression has the asset that the examined problems need not be (approximately) linearly separable in feature space, as is required for successful application of SVMs, but merely differentiable. As a supervised method, it makes more efficient use of the CRP information than reinforcement learning or evolutionary methods [2]. Furthermore, LR is to the knowledge of the authors the only method that can be applied directly to the model of an Arb-PUF and XOR Arb-PUF. Other methods like SVM and Neural Networks build their own intrinsic models.

In our ML experiments, we used an implementation of LR with RProp programmed in our group. The iteration is continued until we reach a point of convergence, i.e., until the averaged prediction rate of two consecutive blocks of five consecutive iterations does not increase anymore for the first time. If the reached performance after convergence on the training set is not sufficient, the process is started anew. After convergence to a good solution on the training set, the prediction error is evaluated on the test set.

2) Evolution Strategies: Evolution Strategies (ES) [1], [44] belong to an ML subfield known as population-based heuristics. They are inspired by the evolutionary adaptation of a population of individuals to certain environmental conditions. In our case, one individual in the population is given by a concrete instantiation of the runtime delays in a PUF, i.e., by a concrete instantiation of the vector $\vec{w}$ appearing in Eqns. 1 and 2. The environmental fitness of the individual is determined by how well it (re-)produces the correct CRPs of the target PUF on a fixed training set of CRPs. ES runs through several evolutionary cycles or so-called *generations*. With a growing number of generations, the challenge-response behavior of the best individuals in the population better and better approximates the target PUF. ES is a randomized method that neither requires an (approximately) linearly separable problem (like Support Vector Machines), nor a differentiable model (such as LR with gradient descent); a merely parameterizable model suffices. Since all known electrical PUFs are easily parameterizable, ES is a very well-suited attack method.

We employed an in-house implementation of ES that is available from our machine learning library PyBrain [43]. The meta-parameters in all applications of ES throughout this paper are (6,36)-selection and a global mutation operator with $\tau = \frac{1}{\sqrt{n}}$. We furthermore used a technique called Lazy Evaluation (LE). LE means that not all CRPs of the training set are used to evaluate an individual's environmental fitness; instead, only a randomly chosen subset is used for evaluation, that changes in every generation. In this paper, we always used subsets of size 2,000 CRPs.

B. Employed Computational Resources

We used three hardware systems to carry out our experiments: A stand-alone, consumer INTEL Quadcore Q9300, and a comparable consumer AMD Quadcore, both worth less than 1,000 Euros. Thirdly, a 30-node cluster of AMD Opteron Quadcores, which represents a worth of around 30,000 Euros. To ensure ease of comparison, all computation times given by us in this paper are calculated for one core of one processor of the corresponding hardware. If k cores are used in parallel, the computation times can be reduced roughly by a factor of $1/k$, since our ML algorithms parallelize straightforwardly.

C. PUF Descriptions and Models

1) Arbiter PUFs: Arbiter PUFs (Arb-PUFs) were first introduced in [17] [23] [46]. It has become standard to describe the functionality of Arb-PUFs via an additive linear delay model [24] [27] [26]. The overall delays of the signals are modeled as the sum of the delays in the stages. In this model, one can express the final delay difference Δ between the upper and the lower path in a k-bit Arb-PUF as $\Delta = \vec{w}^T\vec{\Phi}$, where $\vec{w}$ and $\vec{\Phi}$ are of dimension $k+1$. The parameter vector $\vec{w}$ encodes the delays for the subcomponents in the Arb-PUF stages, whereas the feature vector $\vec{\Phi}$ is solely a function of the applied $k-$bit challenge C [24] [27] [26].

The output t of an Arb-PUF is then determined by the sign of the final delay difference Δ. We make the technical convention of saying that $t = -1$ when the Arb-PUF output is actually 0, and $t = 1$ when the Arb-PUF output is 1:

$$t = sgn(\Delta) = sgn(\vec{w}^T\vec{\Phi}). \quad (3)$$

Eqn. 3 shows that the vector $\vec{w}$ via $\vec{w}^T\vec{\Phi} = 0$ determines a separating hyperplane in the space of all feature vectors $\vec{\Phi}$. Any challenges C that have their feature vector located on the one side of that plane give response $t = -1$, those with feature vectors on the other side $t = 1$. Determination of this hyperplane allows prediction of the PUF.

2) XOR Arbiter PUFs: One possibility to strengthen the resilience of arbiter architectures against machine learning, which has been suggested in [46], is to employ l individual Arb-PUFs in parallel, each with k stages (i.e., each with bitlength k). The same challenge C is applied to all of them, and their individual outputs t_i are XORed in order to produce a global response t_{XOR}. We denote such an architecture as l-XOR Arb-PUF (with the 1-XOR Arbiter PUF being identical to the standard Arbiter PUF).

A formal model for XOR Arb-PUFs can be derived as follows. Making the convention $t_i \in \{-1, 1\}$ as done earlier, it holds that $t_{XOR} = \prod_{i=1}^{l} t_i$. This leads with equation (3) to a parametric model of an l-XOR Arb-PUF, where $\vec{w}_i$ and $\vec{\Phi}_i$ denote the parameter and feature vector, respectively, for the i-th Arb PUF:

$$\begin{aligned} t_{XOR} &= \prod_{i=1}^{l} sgn(\vec{w}_i^T\vec{\Phi}_i) = sgn(\prod_{i=1}^{l} \vec{w}_i^T\vec{\Phi}_i) && (4) \\ &= sgn(\underbrace{\bigotimes_{i=1}^{l} \vec{w}_i^T}_{\vec{w}_{XOR}} \underbrace{\bigotimes_{i=1}^{l} \vec{\Phi}_i}_{\vec{\Phi}_{XOR}}) = sgn(\vec{w}_{XOR}^T\vec{\Phi}_{XOR}) && (5) \end{aligned}$$

While (4) gives a non-linear decision boundary with $l(k+1)$ parameters, (5) defines a linear decision boundary by a separating hyperplane $\vec{w}_{XOR}$ which is of dimension $(k+1)^l$.

3) Lightweight Secure PUFs: Another type of PUF, which we term Lightweight Secure PUF or Lightweight PUF for short, has been introduced in [27]. At its heart are l individual standard Arb-PUFs arranged in parallel, each with k stages (i.e., with bitlength k), which produce l individual outputs $r_1, \ldots, r_l$. These individual outputs are XORed to produce a multi-bit response $o_1, ..., o_m$ of the Lightweight PUF. Another difference to the XOR Arb-PUFs lies in the l inputs $C_1, \ldots, C_l$ which are applied to the l individual Arb-PUFs. Contrary to XOR Arb-PUFs, it does not hold that $C_1 = C_2 = \ldots = C_l = C$, but a more complicated input mapping that derives the individual inputs C_i from the global input C is applied. We refer the reader to [27] for further details.

In order to predict the whole output of the Lightweight PUF, one can apply similar models and ML techniques as in the last section to predict *its single output bits* o_j. While the probability to predict the full output of course decreases exponentially in the misclassification rate of a single bit, the stability of the full output of the Lightweight PUF also decreases exponentially in the same parameters. It therefore seems fair to attack it in the described manner; in any case, our results challenge the bit security of the Lightweight PUF.

4) Feed Forward Arbiter PUFs: Feed Forward Arbiter PUFs (FF Arb-PUFs) were introduced in [17] [23] [24] and further discussed in [26]. Some of their multiplexers are not switched in dependence of an external challenge bit, but as a function of the delay differences accumulated in earlier parts of the circuit. Additional arbiter components evaluate these delay differences, and their output bit is fed into said multiplexers in a "feed-forward loop" (FF-loop). We note that an FF Arb-PUF with k-bit challenges $C = b_1 \cdots b_k$ (i.e., with bitlength k) and l loops has $s = k + l$ multiplexers or stages.

The described dependency makes natural architecture models of FF Arb-PUFs no longer differentiable. Consequently, FF Arb-PUFs cannot be attacked generically with ML methods that require linearly separable or differentiable models (like SVMs or LR), even though such models can be found in special cases, for example for small numbers of non-overlapping loops.

The number of loops as well as the starting and end point of the FF-loops are variable design parameters, and a host of different architectures for an FF Arb-PUF with a moderate or even large number of loops are possible. We conducted first experiments with equally distributed loops that do not overlap (this is the original design suggested in [23]), finding that it was relatively simple to learn. The architecture we eventually investigated in this paper was more resilient to modeling. It consists of loops that are distributed at equal distances over the structure, and which just overlap each other: If the starting point of loop m lies in between stages n and $n + 1$, then the previous loop $m - 1$ has its end point in the immediately following stage $n + 1$. This seemed a natural and straightforward architectural choice; future experiments will have to determine whether this is indeed the optimal (i.e., most secure) architecture.

5) Ring Oscillator PUFs: Ring Oscillator PUFs (RO-PUFs) were discussed in [46], though oscillating loops were proposed in the original silicon PUF paper [15]. While [46] describes the use of Ring Oscillator PUFs in the context of Controlled PUFs and limited-count authentication, it is worth analyzing them as candidate Strong PUFs. A RO-PUF consists of k identically designed ring oscillators, each of which has its own, unique frequency caused by manufacturing variations. The input of a RO-PUF consists of a tuple (i, j), which selects two of the k oscillators. Their frequencies are compared, and the output of the RO-PUF is "0" if the former oscillates faster than the latter, and "1" else. A ring oscillator can be modeled in a straightforward fashion by a tuple of frequencies $(f_1, \ldots, f_k)$. Its output on input (i, j) is "0" if $f_i > f_j$, and "1" else.

D. Numeric CRP Generation, Prediction Error, and Number of CRPs

Given a PUF-architecture that should be examined, the challenge-response pairs (CRPs) that we used in our ML experiments were generated in the following fashion: (i) The delay values for this PUF architecture were chosen pseudo-randomly according to a standard normal distribution. We sometimes refer to this as choosing a certain PUF instance in the paper. (ii) If a response of this PUF instance to a given challenge is needed, the above delays of the two electrical signal paths are simply added up and compared. This methodology follows the well-established linear additive delay model for PUFs [9], [24], [23], [17], [31], [26]. In case of the RO PUF, the frequencies f_i were simply chosen at random according to a normal distribution.

We use the following definitions throughout the paper: The prediction error ϵ is the ratio of incorrect responses of the trained ML algorithm when evaluated on the test set. For all applications of LR, the test set each time consisted of 10,000 randomly chosen CRPs. For all applications of ES (i.e., for the Feed-Forward Arbiter PUF), the test set each time consisted of $8,000$ randomly chosen CRPs. The prediction rate is $1 - \epsilon$.

N_{CRP} (or simply "CRPs") denotes the number of CRPs employed by the attacker in his respective attack, for example in order to achieve a certain prediction rate. This nomenclature holds throughout the whole paper. Nevertheless, one subtle difference should be made explicit: In all applications of LR (i.e., in Sections III to V), N_{CRP} is equal to the size of the training set of the ML algorithm, as one would usually expect. In the applications of ES (i.e., in Section VI), however, the situation is more involved. The attacker needs a test set himself in order to determine which of his many random runs was the best. The value N_{CRP} given in the tables and formulas of Section VI hence reflects the sum of the sizes of the training set and the test set employed by the attacker.

E. FPGA CRP Collection

To obtain CRP data from FPGAs, ten independent instances of Arb-PUFs have been implemented on Spartan-6 FPGAs. The Arb-PUFs were composed of 64 pairs of multiplexers (MUXs) and a D flip-flop based arbiter, and were implemented in Verilog. In order to balance FPGA routing asymmetries, which would otherwise dominate the effect of manufacturing variations, a lookup table (LUT) based Programmable Delay

Line (PDL) has been implemented, as suggested by Majzoobi et al. [28], [29].

We collected 200,000 CRPs from each of our ten FPGA Arb-PUFs instances, resulting in two million CRPs altogether. For each CRP, majority voting over five repetitive measurements of the response to the same challenge was performed in order to determine the final response. For example, if the five measurements resulted in three "0"s and two "1"s, the final response was set to "0". The challenges were generated by a 64-bit pseudo-random number generator (PRNG), which was based on a maximal-length linear feedback shift register (LFSR). The chosen LFSR polynomial generated the maximal-length sequence according to the formula

$$F = 1 + X^1 + X^3 + X^4 + X^{64} \tag{6}$$

where X^n denotes the corresponding 1-bit output from the nth register. This PRNG is cryptographically weak, but it suffices for our purpose of CRP collection, and operates simply and quickly.

F. ASIC CRP Collection

To collect CRPs from ASICs, we built Arb-PUF circuits with 45nm SOI CMOS ASICs. Our Arb-PUF circuits are composed of 64 delay elements and an arbiter circuit element. Each delay element consists of two multiplexers with their inputs connected, leading to 64 pairs of MUXs altogether. The challenge vectors are the select inputs to the MUX pairs, which determine the paths taken by the top and bottom signal, respectively. This leads to 64-bit challenges in our implementation. A SR-latch is used as the arbiter to determine which signal arrived first.

The challenges that we applied to our ASIC Arb-PUFs were generated pseudo-randomly by the same LFSR as in the FPGA case (see Section II-E). To minimize the number of signal IOs on the ASIC PUF test chips, this LFSR was implemented on chip. The LSFR circuit is provided with a "SET" signal and a fixed initial seed, so that it can be reset to a known state when necessary.

40 unpackaged chips of 45nm SOI CMOS technology were taped out for post-silicon measurement. Each chip has two symmetrically placed Arb-PUFs, resulting in 80 PUF instances, 10 of which were used for data collection. To capture the CRPs, we set up a post-silicon validation lab. A microscope station is utilized to mount a 2-pin DC probe and an 8-pin AC probe on the die. Tektronix AFG3252 and Agilent 8251A systems were used to generate "CLK", "SET" and other signals. A PicoScope 5000 with 1GS/s sampling rate is used to capture the response bits. In order to minimize measurement errors, the majority response value of five repetitive measurements was selected as the representative, just as in the case of FPGAs. We captured 200,000 CRPs from each of the ten used PUF instances, resulting in a total of two million CRPs collected from ASICs.

G. PUF Noise and Our Evaluation Methodology

In practice, PUFs may be noisy; but the CRP simulation models used in this paper originally do not incorporate noise. We therefore investigate the ML hardness of the considered Strong PUFs in three different manners.

(1): First, we evaluate the purely "logic" security of PUF designs. Noise-free CRPs from simulations by the additive linear delay model are used in this process. The resulting ML rates indicate the intrinsic security of the considered design.

This security measure has several advantages: Firstly, it is relatively simple to obtain, but still very accurate (see, e.g., Table XI). Secondly, it is independent of any specific PUF implementation and its noise level, as well as of any particular numeric error correction mechanism. Recall that both might change for any new implementation, applications or protocols. Furthermore, the evaluated "logic" security represents an upper limit on a PUF's ML-resilience, at the least in any applications where perfect error correction or fuzzy extractors are utilized to obtain stable responses, such as PUF-based key exchange [11], [4] and oblivious transfer [36], [4] protocols. Finally, the above approach allows a close evaluation of the behavior of the prediction error as a function of the used number of CRPs, the running times of the ML algorithms, the PUF input sizes, and other architectural PUF parameters. Comparably detailed ML experiments on silicon CRPs would require a practically infeasible implementation effort.

One natural side effect of this method is that the obtained prediction errors for the "logic" security of the PUF can lie beyond the stability of a given silicon implementation. This may seem paradoxical at first glance, but is a natural side effect of our approach.

(2): Secondly, we evaluate the performance of ML algorithms on artificially noisy data of the PUF. We do so in a proof of concept for a selected number of architectures and ML methods (see Section VIII). Thereby random noise is injected into the digitally simulated CRP data by inverting a certain percentage of all (single-bit) PUF outputs. The outputs to be flipped are chosen uniformly at random.

This approach gives a general indication of the error-tolerance of the ML algorithms. The uniform choice of flipped responses is no optimal noise model from a circuit perspective. But the approach realistically describes situations in which the attacker is limited to eavesdropping a noisy channel for collecting his PUF-CRPs. This situation practically occurs in PUF protocol eavesdropping, or if malware transfers PUF CRPs to the adversary. It also accurately models situations where noisy and error-prone digital probing is used to collect the PUF-CRPs (compare Section I-B).

In order to stay close to this attack model, the prediction error is evaluated on a set of noise-free CRPs. This allows us to better isolate and quantify the effect that noise has on the prediction quality. Similar to above, this has the natural consequence that the achieved prediction error can be smaller than the injected noise level.

(3): Finally, we evaluate the feasibility of our attacks on real, silicon systems, again in selected proof of concept experiments (see Section IX). We assume that the adversary has got physical access to the Strong PUF and its public CRP interface, as it is common in the established PUF attack model. He can thus repeat CRP measurements at will in order to gain output stability, or put the PUF to chosen ambient conditions

that ensure particular reliability.

We carried out our proof of concept attacks on Arbiter PUFs and XOR Arbiter PUFs, both on FPGAs and ASICs CRPs, keeping the PUF at the same room temperature and using majority voting over several measurements (compare Sections II-E and II-F). Again, this allowed us to derive extremely acccurate models, whose predication rate (for these fixed ambient conditions) is better than the general stability of the PUF over the entire temperature range of its potential use. The occurence of this phenomenon in real silicon systems finally confirms its non-paradoxical nature.

The results we obtained throughout this paper in steps (1), (2) and (3) are very close to each other. Among other things, our work therefore establishes the high suitability of the "logic" hardness of a PUF as a measure for the PUF's general security, at the least for our considered class of delay-based PUFs.

Closely related to the above discussion is the question when a modeling attack on a PUF should be called successful in practice. Given our above discussion, the following criteria appear suggestive: If the security of a concrete PUF implementation is considered as in step (3), the attack should be called successful if the achieved prediction rate is better than the stability of this PUF within the temperature, voltage and aging variations envisaged during its use. Dependent on the exact attack model, the CRPs for the attack thereby may be measured under ambient conditions controlled by the adversary. For example, measures such as repeated measurements and majority voting may be allowed to stabilize the output.

If the security of an abstract PUF design is evaluated, as in steps (1) and (2), the attack can be called successful if it significantly exceeds the realistic stability levels of currently existing implementations, even though this criterion is somewhat vague. Another abstract criterion, which is sufficient but not necessary, is the growth rate: If the preduction error is related linearly or low-degree polynomially to the PUF's challenge length, its architectural parameters and the number of CRPs used in the ML experiment, an abstract PUF design should no longer be called secure.

III. Arbiter PUFs

We now start the results part of the paper by presenting our findings for standard Arbiter PUFs on simulated, noise-free data.

A. Machine Learning Results

To determine the separating hyperplane $\vec{w}^T\vec{\Phi} = 0$, we applied SVMs, LR and ES. LR achieved the best results, which are shown in Table I. We chose three different prediction rates as targets: 95% is roughly the environmental stability of a 64-bit Arbiter PUF when exposed to a temperature variation of 45C and voltage variation of $\pm 2\%$ [4]. The values 99% and 99.9%, respectively, represent benchmarks for optimized ML results. All figures in Table I were obtained by averaging over 5 different training sets. Accuracies were estimated using test sets of 10,000 CRPs.

[4]The exact figures reported in [24] are: 4.57% CRP variation for a temperature variation of 45C, and 2.16% for a voltage variation of $\pm 2\%$.

ML Method	Bit Length	Prediction Rate	CRPs	Training Time
LR	64	95%	640	0.01 sec
		99%	2,555	0.13 sec
		99.9%	18,050	0.60 sec
LR	128	95%	1,350	0.06 sec
		99%	5,570	0.51 sec
		99.9%	39,200	2.10 sec

TABLE I
LR on Arbiter PUFs with 64 and 128 stages (i.e., with bitlength 64 and 128), for noise-free, simulated CRPs.

B. Scalability

We also executed scalability experiments with LR, which are displayed in Figure 1 and Figure 2. They show that the relevant parameters – the required number of CRPs in the training set and the computational complexity, i.e., the number of basic operations – grow linearly or low-degree polynomially in the misclassification rate ϵ and the length k of the Arb PUF. Theoretical considerations (dimension of the feature space, Vapnik-Chervonenkis dimension [3]) suggest that the *minimal* number of CRPs N_{CRP} that is necessary to model a k-stage arbiter with a misclassification rate of ϵ should obey the relation

$$N_{CRP} = O(k/\epsilon). \tag{7}$$

This was confirmed by our experimental results.

In practical PUF applications, it is essential to know the concrete number of CRPs that may become known before the PUF-security breaks down. Assuming an approximate linear functional dependency $y = ax + c$ in the double logarithmic plot of Figure 1 with a slope of $a = -1$, we obtained the following empirical formula (8). It gives the approximate number of CRPs N_{CRP} that is required to learn a k-stage arbiter PUF with error rate ϵ:

$$N_{CRP} \approx 0.5 \cdot \frac{k+1}{\epsilon} \tag{8}$$

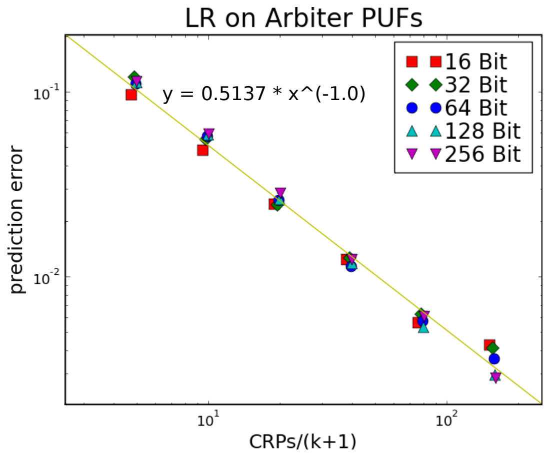

Fig. 1. Double logarithmic plot of misclassification rate ϵ on the ratio of training CRPs N_{CRP} and $dim(\Phi) = k + 1$.

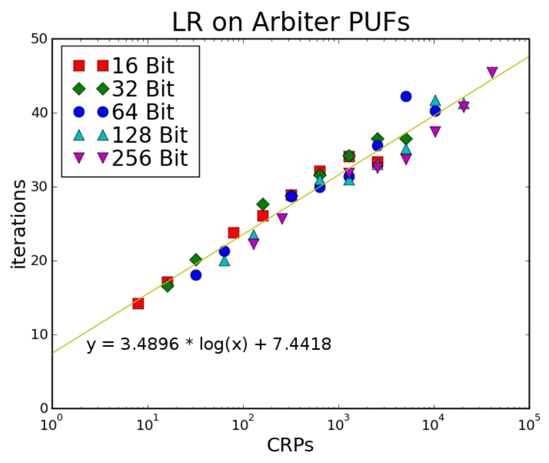

Fig. 2. No. of iterations of the LR algorithm until "convergence" occurs (see section II), plotted in dependence of the training set size N_{CRP}.

Our experiments also showed that the *training time of the ML algorithms*, measured in terms of the number of basic operations N_{BOP}, grows slowly. It is determined by the following two factors: (i) The evaluation of the current model's likelihood (Eqn. 1) and its gradient (Eqn. 2), and (ii) the number of iterations of the optimization procedure before convergence occurs (see section II-A1). The former is a sum over a function of the feature vectors $\vec{\Phi}$ for all N_{CRP}, and therefore has complexity $O(k \cdot N_{CRP})$. On the basis of the data shown in Figure 2, we may further estimate that the numbers of iterations increases proportional to the logarithm of the number of CRPs N_{CRP}. Together, this yields an overall complexity of

$$N_{BOP} = O\left(\frac{k^2}{\epsilon} \cdot \log \frac{k}{\epsilon}\right). \tag{9}$$

IV. XOR Arbiter PUFs

We continue by examining XOR Arbiter PUFs on simulated, noise-free CRPs.

A. Machine Learning Results

In the application of SVMs and ES to XOR Arb-PUFs, we were able to break small instances, for example XOR Arb-PUFs with 2 or 3 XORs and 64 stages. LR significantly outperformed the other two methods. The key observation is that instead of determining the linear decision boundary (Eqn. 5), one can also specify the non-linear boundary (Eqn. 4). This is done by setting the LR decision boundary $f = \prod_{i=1}^{l} \vec{w}_i^T \vec{\Phi}_i$. The results are displayed in Table II.

B. Scalability

Figures 3 and 4 display the results of our scaling experiments with LR. Again, the *smallest* number of CRPs in the training set N_{CRP} needed to achieve predictions with a misclassification rate ϵ scales linearly with the number of parameters of the problem (the product of the number of stages k and the number of XORed Arb-PUFs l):

$$N_{CRP} \sim \frac{(k+1) \cdot l}{\epsilon}. \tag{10}$$

But, in contrast to standard Arb-PUFs, optimizing the non-linear decision boundary (4) on the training set now is a non-convex problem, so that the LR algorithm is not guaranteed to find (an attractor of) the global optimum in its first trial. It needs to be iteratively restarted N_{trial} times. N_{trial} thereby can be expected to not only depend on k and l, but also on the size N_{CRP} of the employed training set.

As is argued in greater detail in [45], the success rate ($= 1/N_{trial}$) of finding (an attractor of) the global optimum is determined by the ratio of dimensions of gradient information ($\propto N_{CRP}$ as the gradient is a linear combination of the feature vector) and the dimension d_Φ in which the problem is linear separable. The dimension d_Φ is the number of independent dimensions of $\vec{\Phi}_{XOR} = \bigotimes_{i=1}^{l} \vec{\Phi}_i = \bigotimes_{i=1}^{l} (\Phi_i^1 \ldots, \Phi_i^k, 1)^T$.

As the tensor product of several vectors consists of all possible products between their vector components, the independent dimensions are given by the number of different products of the form $\Phi_1^{i_1} \cdot \Phi_2^{i_2} \cdot \ldots \Phi_l^{i_l}$ for $i_1, i_2, \ldots, i_l \in \{1, 2, \ldots, k+1\}$ (where we say that $\Phi_i^{k+1} = 1$ for all $i = 1, \ldots, l$). For XOR Arb-PUFs, we furthermore know that the same challenge is applied to all l internal Arbiter PUFs, which tells us that $\Phi_j^i = \Phi_{j'}^i = \Phi^i$ for all $j, j' \in \{1, \ldots, l\}$ and $i \in \{1, \ldots, k+1\}$. Since a repetition of one component does not affect the product regardless of its value (recall that $\Phi^r \cdot \Phi^r = \pm 1 \cdot \pm 1 = 1$), the number of the above products can be obtained by counting the unrepeated components. The number of different products of the above form is therefore given as the number of l-tuples without repetition, plus the number of $(l-2)$-tuples without

ML Method	Bit Length	Pred. Rate	No. of XORs	CRPs ($\times 10^3$)	Training Time
LR	64	99%	4	12	3:42 min
			5	80	2:08 hrs
			6	200	31:01 hrs
LR	128	99%	4	24	2:52 hrs
			5	500	16:36 hrs
			6	—	—

TABLE II
LR on XOR Arbiter PUFs for noise-free, simulated CRPs. Training times are averaged over different PUF-instances.

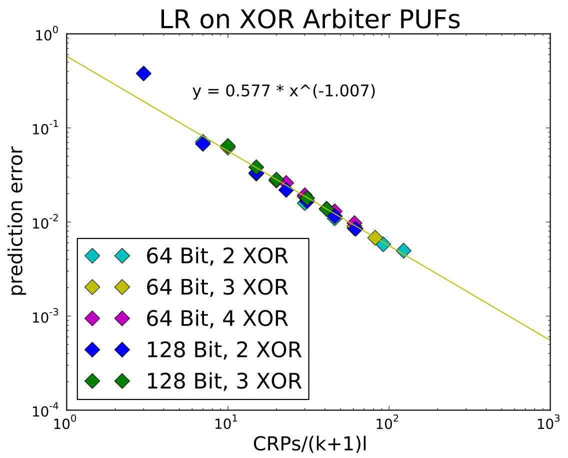

Fig. 3. Double logarithmic plot of misclassification rate ϵ on the ratio of training CRPs N_{CRP} and problem size $dim(\Phi) = (k+1) \cdot l$.

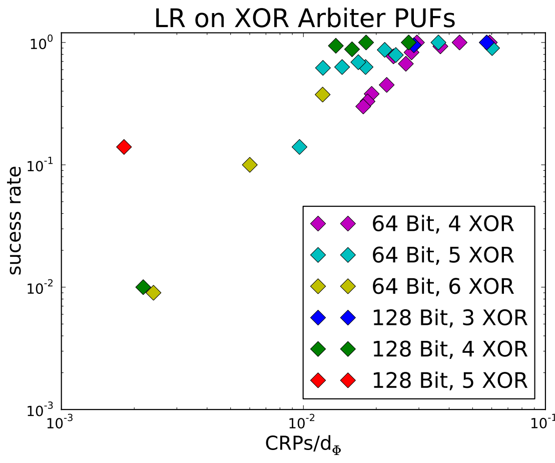

Fig. 4. Average rate of success of the LR algorithm plotted in dependence of the ratio d_Φ (see Eqn. (11)) to N_{CRP}.

repetition (corresponding to all l-tuples with 1 repetition), plus the number of $(l-4)$-tuples without repetition (corresponding to all l-tuples with 2 repetitions), etc.

Writing this down more formally, d_Φ is given by

$$\begin{aligned} d_\Phi &= \binom{k+1}{l} + \binom{k+1}{l-2} + \binom{k+1}{l-4} + \dots \\ &\overset{k \gg l}{\approx} \frac{(k+1)^l}{l!}. \end{aligned} \tag{11}$$

The approximation applies when k is considerably larger than l, which holds for the considered PUFs for stability reasons. Following [45], this seems to lead to an expected number of restarts N_{trial} to obtain a valid decision boundary on the training set (that is, a parameter set $\vec{w}$ that separates the training set), of

$$N_{trial} = O\left(\frac{d_\Phi}{N_{CRP}}\right) = O\left(\frac{(k+1)^l}{N_{CRP} \cdot l!}\right). \tag{12}$$

Furthermore, each trial has the complexity

$$T_{trial} = O\left(\,(k+1) \cdot l \cdot N_{CRP}\,\right). \tag{13}$$

V. Lightweight Secure PUFs

This section investigates the ML-resilience of LW PUFs on simulated, noise-free CRPs.

A. Machine Learning Results

In order to test the influence of the specific input mapping of the Lightweight PUF on its machine-learnability (see Sec. II-C), we examined architectures with the following parameters: variable l, $m = 1$, $x = l$, and arbitrary s. We focused on LR right from the start, since this method was best in class for XOR Arb-PUFs, and obtained the results shown in Table III. The specific design of the LW PUF leads to significantly increased training times and CRP requirements. Still, we were able to predict single output bits for LW PUFs with up to 5 XORs with probabilities of 99%, both for bit lengths 64 and 128 bits.

Bit Length	Pred. Rate	No. of XORs	CRPs	Training Time
64	99%	3	6,000	8.9 sec
		4	12,000	1:28 hrs
		5	300,000	13:06 hrs
128	99%	3	15,000	40 sec
		4	500,000	59:42 min
		5	10^6	267 days

TABLE III
LR on Lightweight PUFs for noise-free, simulated CRPs. Prediction rate refers to single output bits. Training times were averaged over different PUF instances.

B. Scalability

Some theoretical consideration [45] shows the underlying ML problem for the Lightweight PUF and the XOR Arb PUF are similar with respect to the required CRPs, but differ quantitatively in the resulting runtimes. The asymptotic formula on N_{CRP} given for the XOR Arb PUF (Eqn. 10) analogously also holds for the Lightweight PUF. But due to the influence of the special challenge mapping of the Lightweight PUF, the number N_{trial} has a growth rate that is different from Eqn. 12. It seems to lie between $O(\frac{(k+1)^l}{N_{CRP} \cdot l!}))$ and the related expression $O(\frac{(k+1)^l}{N_{CRP}})$ [45]. While these two formulas differ by factor of $l!$, we note that in our case $k \gg l$, and that l is comparatively small for stability reasons. Again, all these considerations on N_{CRP} and N_{Trial} hold for the prediction of single output bits of the Lightweight PUF.

These points were at least qualitatively confirmed by our scalability experiments. We observed agreement with the above discussion in that with the same ratio $CRPs/d_\Phi$ the LR algorithm will have a longer runtime for the Lightweight PUF than for the XOR Arb-PUF. For example, while with a training set size of $12,000$ for the 64-bit 4-XOR Arb-PUF on average about 5 trials were sufficient, for the corresponding Lightweight PUF 100 trials were necessary.

VI. Feed Forward Arbiter PUFs

We consider the case of FF Arb-PUFs on simulated, noise-free CRPs in this section.

A. Machine Learning Results

Recall from Section II-C4 that FF Arb-PUFs with k-bit challenges $C = b_1 \cdots b_k$ (i.e., with bitlength k) and l loops have $s = k + l$ multiplexers or stages. We experimented with SVMs and LR on these PUFs, using different models and input representations, but could only break special cases with small numbers of non-overlapping FF loops, such as $l = 1, 2$. This is in agreement with earlier results reported in [26].

The application of ES finally allowed us to tackle much more complex FF-architectures with up to 8 FF-loops. In the architectures examined by us, all loops have equal length, and are distributed regularly over the PUF, with overlapping start- and endpoints of successive loops, as described in Section II-C. Table IV shows the results we obtained. The given prediction rates are the best of 40 trials on one randomly chosen PUF-instance of the respective length. The given CRP numbers are

Bit Length	FF-loops	Pred. Rate Best Run	CRPs	Training Time
64	6	97.72%	50,000	07:51 min
	7	99.38%	50,000	47:07 min
	8	99.50%	50,000	47:07 min
	9	98.86%	50,000	47:07 min
	10	97.86%	50,000	47:07 min
128	6	99.11%	50,000	3:15 hrs
	7	97.43%	50,000	3:15 hrs
	8	98.97%	50,000	3:15 hrs
	9	98.78%	50,000	3:15 hrs
	10	97.31%	50,000	3:15 hrs

TABLE IV
ES ON FEED-FORWARD ARBITER PUFS FOR NOISE-FREE, SIMULATED CRPS. PREDICTION RATES ARE FOR THE BEST OF A TOTAL OF 40 TRIALS ON A SINGLE, RANDOMLY CHOSEN PUF INSTANCE. TRAINING TIMES ARE FOR A SINGLE TRIAL. WE APPLIED LAZY EVALUATION WITH 2,000 CRPS.

the sum of the training set and the test set employed by the attacker; a fraction of 5/6 was used as the training set, 1/6 as the test set (see Section II-D). We note for comparison that in-silicon implementations of 64-bit FF Arb-PUFs with 7 FF-loops are known to have an environmental stability of 90.16% [24].

B. Scalability

We started by empirically investigating the CRP growth as a function of the number of challenge bits, examining architectures of varying bitlength that all have 6 FF-loops. The loops are distributed as described in Section II-C. The corresponding results are shown in Figure 5. Every data point corresponds to the averaged prediction error of 10 trials on the same, random PUF-instance.

Secondly, we investigated the CRP requirements as a function of a growing number of FF-loops, examining architectures with 64 bits. The corresponding results are depicted in Figure 6. Again, each data point shows the averaged prediction error of 10 trials on the same, random PUF instance.

In contrast to the Sections IV-B and V-B, it is now much more difficult to derive reliable scalability formulas from this data. The reasons are threefold. First, the structure of ES provides less theoretical footing for formal derivations. Second, the random nature of ES produces a very large variance in the data points, making also clean empirical derivations more difficult. Third, we observed an interesting effect when comparing the performance of ES vs. SVM/LR on the Arb PUF: While the supervised ML methods SVM and LR showed a linear relationship between the prediction error ϵ and the required CRPs even for very small ϵ, ES proved more CRP hungry in these extreme regions for ϵ, clearly showing a superlinear growth. The same effect can be expected for FF architectures, meaning that one consistent formula for extreme values of ϵ may be difficult to obtain.

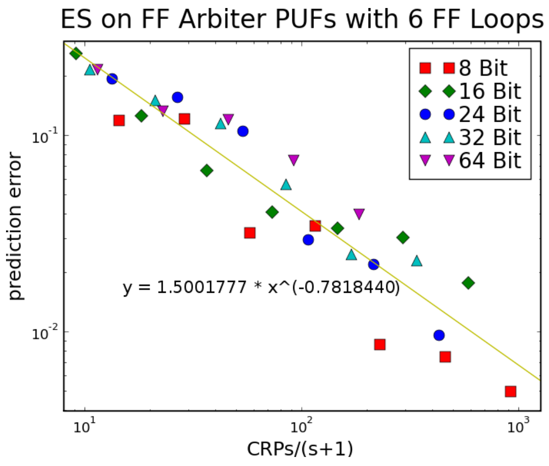

Fig. 5. Results of 10 trials per data point with ES for different lengths of FF Arbiter PUFs and the hyperbola fit.

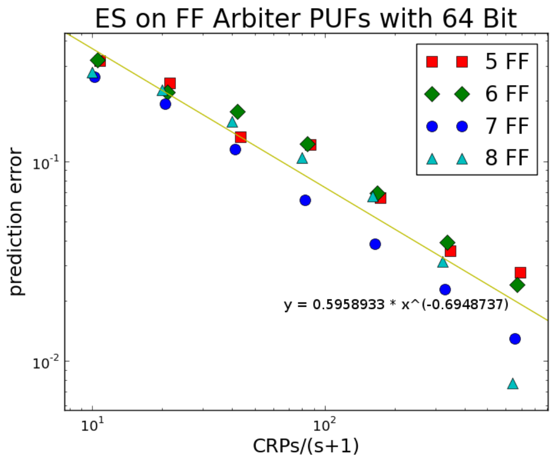

Fig. 6. Results of 10 trials per data point with ES for different numbers of FF-loops and the hyperbola fit.

It still seems somewhat suggestive from the data points in Figures. 5 and 6 to conclude that the growth in CRPs is about linear, and that the computation time grows polynomially. For the reasons given above, however, we would like to remain conservative, and present the upcoming empirical formulas only in the status of a conjecture.

The data gathered in our experiments is best explained by assuming a qualitative relation of the form

$$N_{CRP} = O(s/\epsilon^c) \tag{14}$$

for some constant $0 < c < 1$, where s is the number of stages in the PUF. Concrete estimation from our data points leads to an approximate formula of the form

$$N_{CRP} \approx 9 \cdot \frac{s+1}{\epsilon^{3/4}}. \tag{15}$$

The *computation time* required by ES is determined by the following factors: (i) The computation of the vector product $\vec{w}^T\vec{\Phi}$, which grows linearly with s. (ii) The evolution applied to this product, which is negligible compared to the other steps. (iii) The number of iterations or *"generations"* in ES until a small misclassification rate is achieved. We conjecture that this grows linearly with the number of multiplexers s. (iv) The number of CRPs that are used to evaluate the individuals per iteration. If Eqn. 15 is valid, then N_{CRP} is on the order of $O(s/\epsilon^c)$.

Assuming the correctness of the conjectures made in this derivation, this would lead to a polynomial growth of the computation time in terms of the relevant parameters k, l and s. It could then be conjectured that the number of basic computational operations N_{BOP} obeys

$$N_{BOP} = O(s^3/\epsilon^c) \tag{16}$$

for some constant $0 < c < 1$.

VII. Ring Oscillator PUFs

Ring Oscillator PUFs (RO PUFs) to some extent constitute an exception within this paper. They are a relatively versatile PUF structure, and have been suggested for uses in various contexts and also under different specific designs. The majority of these suggestions applies them the context of Weak PUFs or Controlled PUFs, i.e., in applications where their CRP interface is not publicly accessible for external parties. One typical example would be their use within pseudo-random digital number generators which employ the RO-responses as a secret seed. We stress once more that our modeling attacks apply in such application contexts either not at all, or only under very rare and restricted circumstances; compare again our discussion in Section I-B.

Still, in order to complete our picture on delay-based PUFs, it seems worthwhile to clarify the security of the RO PUF if it is used as Strong PUF, i.e, if its CRP interface can be accessed without restrictions, or if its CRP are sent in the clear in protocols and can be eavesdropped. The specific type of ring oscillator PUF we analyse is taken from [46]: It employs k ring oscillators overall. Two of them are selected by a challenge, and their frequencies are compared in order to produce a single output bit. This structure leads to $k(k-1)/2$ possible challenges.

There are several strategies to attack this particular type of RO-PUF if it is used as a Strong PUF. A first, straightforward attempt would be a simple collection or read out of all its (quadratically many) $k(k-1)/2$ CRPs.

A more interesting case is if Eve can choose the CRPs adaptively. This case occurs if the CRP interface is public and she has physical access to it. She can then improve her attack, employing a standard sorting algorithm to obtain the RO-PUF's frequencies $(f_1, \dots, f_k)$ in ascending order. This strategy subsequently allows her to predict the outputs without knowing the exact frequencies f_i themselves. The time and CRP complexities of the respective sorting algorithms are well known [32]; for example, there are several algorithms with average- and even worst-case CRP complexity of $N_{CRP} = O(k \cdot \log k)$. Their running times are also low-degree polynomial.

ML Method	No. of Oscill.	Pred. Rate Average		CRPs	
QS	256	99%	99.9%	14,060	28,891
	512	99%	99.9%	36,062	103,986
	1024	99%	99.9%	83,941	345,834

TABLE V
QUICK SORT APPLIED TO THE RING OSCILLATOR PUF. THE GIVEN CRPS ARE AVERAGED OVER 40 TRIALS.

Perhaps the most advanced case is when Eve cannot adaptively choose the CRPs, but is restricted to *eavesdropped* CRPs, which were chosen randomly by other parties. We carried out experiments for this case, in which we applied Quick Sort (QS) to randomly drawn CRPs. The results are shown in Table V. The estimated required number of CRPs is given by

$$N_{CRP} \approx \frac{k(k-1)(1-2\epsilon)}{2+\epsilon(k-1)}, \tag{17}$$

and the training times are low-degree polynomial. Among other things, Eqn. 17 quantifies for how many runs RO-PUFs can be used in identification protocols à la Pappu et al. [33], [34], even under the assumption that the adversary is limited to CRP eavesdropping and never can access the PUF physically.

CRPs ($\times 10^3$)		Percentage of error-inflicted CRPs			
		0%	2%	5%	10%
24	Best Pr.	98.76%	92.83%	88.05%	—
	Ave. Pr.	98.62%	91.37%	88.05%	—
	Suc. Tr.	0.6%	0.8%	0.2%	0.0%
	Conv.	40.0%	25.0%	5.0%	0.0%
50	Best Pr.	99.49%	95.17%	92.67%	89.89%
	Ave. Pr.	99.37%	94.39%	91.62%	88.20%
	Suc. Tr.	12.4%	13.9%	10.0%	4.6%
	Conv.	100.0%	62.5%	50.0%	20.0%
200	Best Pr.	99.88%	97.74%	96.01%	94.61%
	Ave. Pr.	99.78%	97.34%	95.69%	93.75%
	Suc. Tr.	100.0%	87.0%	87.0%	71.4%
	Conv.	100.0%	100.0%	100.0%	100.0%

TABLE VI
LR ON 128-BIT, 4-XOR ARB PUFS WITH DIFFERENT LEVELS OF NOISE IN THE TRAINING SET AND NOISE-FREE TEST SETS. WE SHOW THE BEST AND AVERAGE PREDICTION RATES OF 40 RANDOMLY CHOSEN INSTANCES, THE PERCENTAGE OF SUCCESSFUL TRIALS OVER THESE INSTANCES, AND THE PERCENTAGE OF INSTANCES THAT CONVERGED TO A SUFFICIENT OPTIMUM IN AT LEAST ONE TRIAL.

VIII. Results on Error-Inflicted CRPs

Having examined the performance of ML algorithms on simulated, noise-free CRPs over the last sections, we now investigate the effect of noise and errors in the CRPs. For various noise levels, we choose an fraction of all CRPs uniformly at random, and flip their single-bit responses. We then run the ML algorithm on the noise-inflicted data, and evaluate its performance on a noise-free training set. This allows us to precisely pinpoint the effect of the erroneous CRPs. For a further discussion on our methodology, please see Section II-G. Our findings were that our ML algorithms are very robust with respect to the examined error levels. This again confirms the relevance and validity of the purely "logic" ML hardness as a measure for PUF security; compare again our discussion in Section II-G.

A. LR on XOR Arbiter PUFs with Error-Inflicted CRPs

We started by investigating LR on XOR Arbiter PUFs. The results are displayed in Tables VI and VII for various noise levels. They show that LR can cope very well with errors, provided that around three to four times more CRPs are used. The required convergence times on error inflicted training sets did not change substantially compared to error free training sets of the same sizes.

CRPs ($\times 10^3$)		Percentage of error-inflicted CRPs			
		0%	2%	5%	10%
500	Best Pr.	99.90%	97.55%	96.48%	93.12%
	Ave. Pr.	99.84%	97.33%	95.84%	93.12%
	Suc. Tr.	7.0%	2.9%	0.9%	0.7%
	Conv.	20.0%	20.0%	10.0%	5.0%

TABLE VII
LR ON 128-BIT, 5-XOR ARB PUFS WITH DIFFERENT AMOUNTS OF ERROR IN THE TRAINING SET. REST AS IN THE CAPTION OF TABLE VI.

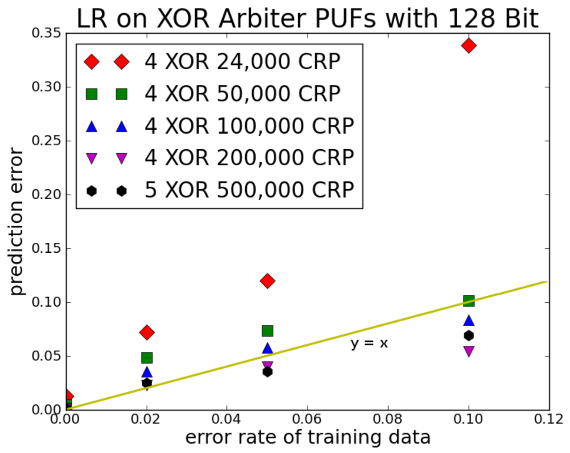

Fig. 7. Graphical illustration of the effect of error on LR in the training set, with chosen data points from Tables VI and VII.

B. ES on Feed-Forward Arbiter PUFs with Error-Inflicted CRPs

In the same manner as above, we investigated the performance of ES on FF Arb PUFs when it is run with error-inflicted CRPs. The results are shown in Table VIII and Figure 8. ES possesses an extremely high tolerance against the inflicted errors; its performance is hardly changed at all.

IX. Results on Silicon CRPs

So far, all of our results were achieved on numerically simulated CRPs. In any simulations of the Arbiter PUF variants, the additive linear delay model has been used (see Section II-C). Based on earlier experiments with silicon implementations [24], [9], it had been conjectured in the first version of this work that this model is accurate enough that our attacks transfer well to the silicon case [41].

We are now able to conduct a detailed validation of this conjecture, both for ASIC and FPGA implementations, in this section. The two architectures we chose to investigate were Arbiter PUFs and XOR Arbiter PUFs. They are the two most relevant designs in our context: For RO PUFs, the analytical model, which simply assigns one frequency to each oscillator, is very close to reality. FF Arb PUFs and and Lightweight PUFs are also delay-based, therefore it can be assumed that our results on (XOR) Arb PUFs transfer well to their case. In our analysis, we used overall more than four million silicon CRPs from FPGAs and ASICs (see Sections II-F and II-E).

CRPs ($\times 10^3$)		Percentage of error-inflicted CRPs			
		0%	2%	5%	10%
50	Best Pr.	98.29%	97.78%	98.33%	97.68%
	Ave. Pr.	89.94%	88.75%	89.09%	87.91%
	Suc. Tr.	42.5%	37.5%	35.0%	32.5%

TABLE VIII
ES ON 64-BIT, 6 FF ARB PUFS WITH DIFFERENT LEVELS OF NOISE IN THE TRAINING SET AND NOISE-FREE TEST SETS. WE SHOW THE BEST AND AVERAGE PREDICTION RATES FROM OVER 40 INDEPENDENT TRIALS ON A SINGLE, RANDOMLY CHOSEN PUF INSTANCE, AND THE PERCENTAGE OF SUCCESSFUL TRIALS THAT CONVERGED TO 90% OR BETTER.

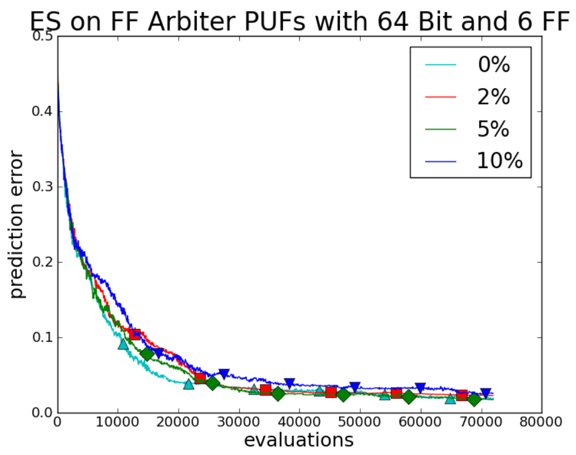

Fig. 8. Graphical illustration of the tolerance of ES to errors. We show the best result of 40 independent trials on one randomly chosen PUF instance for varying error levels in the training set. The results hardly differ.

For standard Arbiter PUFs, the CRP-stability of the used FPGA systems (again under majority voting) was at 95.13% under an artificially injected ±5% voltage variation. For ASICs, this number was 96.82%. These figures also give us an indication of the projected stability of the two systems under varying temperature and aging, even though we did not execute detailed studies on the latter two. Interestingly, our obtained ML prediction rates exceeded these noise levels. The reason is that we assumed realistically in our measurements that an attacker with physical access could collect the CRPs at one single, relatively stable temperature level, and could apply majority voting to stabilize the responses (compare Section II-G).

Overall, the findings detailed in the next subsections confirm that there is little perfomance loss of our method for silicon CRPs. This establishes the good applicability of the linear additive delay model in any future security analyses of delay-based PUFs, and again confirms our paradigm that the performance on noise-free, simulated CRPs is a very good indicator for a delay-based PUF's security. It turned out in this context that FPGA-CRPs were slightly harder to learn than the ASIC data. Two conceivable causes could be the slightly higher noise levels of FPGAs (see above), and the insertion of PDLs (Programmable Delay Lines) on FPGAs, which makes the MUX structures more complicated.

A. Results on Silicon Arbiter PUFs

As described in detail in Sections II-E and II-F, we used ten PUF-instances on FPGAs and ten on ASICs, and collected 200,000 CRPs of each of them, applying majority voting on five responses for each challenge. Table IX gives the results of

ML Method	CRP Source	Prediction Rate	CRPs	Training Time
LR	FPGA	$> 95\%$	650	0.12 sec
		$> 99\%$	6500	0.83 sec
LR	ASIC	$> 95\%$	650	0.11 sec
		$> 99\%$	6500	0.76 sec

TABLE IX
LR ON ARB PUFS OF BITLENGTH 64 FOR FPGA AND ASIC DATA, COLLECTED UNDER STABLE TEMPERATURE AND MAJORITY VOTING.

our LR algorithm on the FPGA and ASIC data, respectively. They are very close to the earlier findings for synthetic CRPs (see Section III and Table I). Only for very small prediction errors slightly below 1%, the known small deviations from the linear additive delay model, possible measurement errors, and instabilities come into play and have a notable effect. This makes it more difficult to achieve extremely low prediction rates significantly below 1%; a strongly increasing amount of CRP data is required for such low rates. Anyway, in practice a prediction error of 1% or below is already sufficient to break the system; compare the stability levels mentioned above.

1) Scalability: Similar to Section III-B, we conducted scaling experiments on FPGA and ASIC data. We investigated the relationship between the number of CRPs and prediction rates, as well as the overall running time of our algorithm.

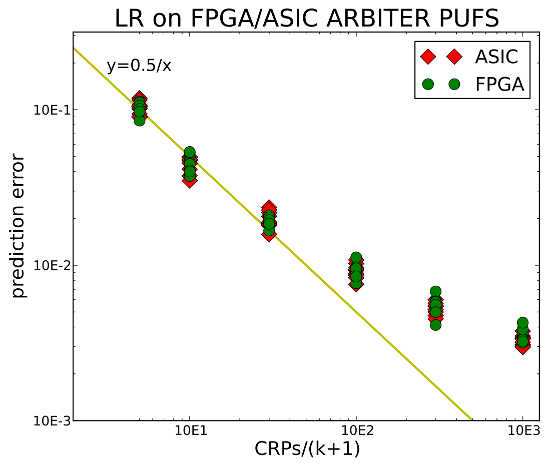

Fig. 9. Performance of LR on FPGA and ASIC Arbiter PUFs for small prediction errors. Each data point represents a single PUF instance.

Figure 9 depicts the results of our scaling experiments on the required number of CRPs for FPGA and ASIC data. The figure shows that the linear relation of Section III-B between the number of CRPs and the prediction rate holds very well for a prediction error of above 1%. In this regime, it is described by exactly the same formula as in Section III-B:

$$N_{CRP} \approx 0.5 \cdot \frac{k+1}{\epsilon}. \tag{18}$$

Below around 1%, a saturation effect occurs, however. Reducing the prediction error further is still possible, but increasingly requires more than a linear number of CRPs. In this regime, the limits of the additive linear delay model begin to show. Possible measurement errors and instabilities contribute to this phenomenon, too.

Interestingly, this effect concerns FPGAs and ASICs in exactly the same fashion. Among other things, this confirms that Majzoobi et al.'s method of balancing the routing asymmetries of FPGAs via look-up tables [28], [29] works very well (see Section II-E).

The second aspect we investigated is the scaling of the overall runtime of our algorithm. It is given in Figure 10. Our results can be seen as confirmation that the basic relationship given in Section IV-B still holds, and that the runtime scales as

$$N_{BOP} = O\left(\frac{k^2}{\epsilon} \cdot \log \frac{k}{\epsilon}\right). \tag{19}$$

Still, some differences between the silicon and simulated CRPs regarding are observable; noise and deviations from the perfect linear additive delay model have a stronger effect in the XOR case than in the case of single Arb-PUFs, and increase the training times.

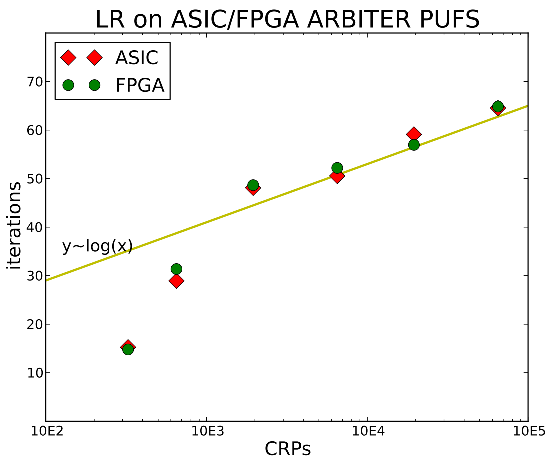

Fig. 10. Necessary trials for LR on FPGA and ASIC Arbiter PUFs.

B. Results on Silicon XOR Arbiter PUFs

We also investigated the case of XOR Arbiter PUFs for FPGA and ASIC data. Our results are summarized in Table X. Again, they are relatively close to our earlier findings of Section IV-A. However, the small deviations from the linear additive delay model now certainly have a stronger effect, since we consider the XOR of several single Arbiter PUFs. We were not able to learn 6-XOR Arb PUFs anymore with the collected amount of data. Extrapolating from our previous experience, we believe that about 700,000 CRPs would be necessary to this end.

1) Scalability: We also conducted detailed scalability experiments, following the methodology of Section IV-B. The required number of CRPs vs. the achieved prediction error is shown in Figure 11. It shows that for XOR Arb PUFs, the saturation effect is similar to single Arbiter PUFs. The only difference is that it already starts at slightly lower prediction rates, and slowly increases with the number of XORs. Still, the saturation is so mild that also prediction errors below 1%

ML Method	CRP Source	Pred. Rate	No. of XORs	CRPs ($\times 10^3$)	Training Time
LR	FPGA	$> 99\%$	3	19.5	51.5 sec
			4	39	139 sec
			5	78	39 min
LR	ASIC	$> 99\%$	3	19.5	26 sec
			4	39	63.5 sec
			5	78	18:09 min

TABLE X
LR ON XOR ARB PUFS OF BITLENGTH 64 FOR FPGA AND ASIC DATA (COLLECTED UNDER STABLE TEMPERATURE AND MAJORITY VOTING). TRAINING TIMES ARE AVERAGED OVER DIFFERENT PUF-INSTANCES.

can be achieved, provided that a sufficient amount of CRPs is used. Over 1%, the basic relationship

$$N_{CRP} \sim \frac{(k+1)\cdot l}{\epsilon}. \tag{20}$$

appears to hold well, as discussed already in Section IV-B.

In terms of computation times, our findings are summarized in Figure 12. It corresponds to Figure 4 in Section IV-B, which used simulated CRPs. Again, our results at least qualitatively confirm the scaling behavior we earlier oberserved on simulated data. Also for FPGA and ASIC data, the expected number of restarts N_{trial} to obtain a valid decision boundary on the training set (that is, a parameter set $\vec{w}$ that separates the training set), is given approximately by

$$N_{trial} = O\left(\frac{d_\Phi}{N_{CRP}}\right) = O\left(\frac{(k+1)^l}{N_{CRP}\cdot l!}\right). \tag{21}$$

Furthermore, each trial again has the approximate complexity

$$T_{trial} = O\left((k+1)\cdot l \cdot N_{CRP}\right). \tag{22}$$

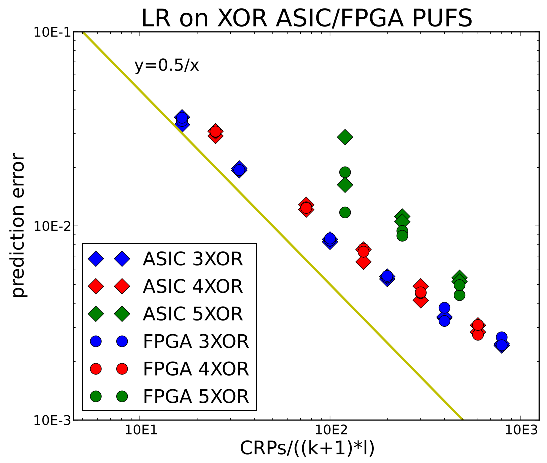

Fig. 11. Performance of LR on XOR Arbiter PUFs for FPGA and ASIC data for small prediction errors.

X. Summary

A. Summary

We investigated the resilience of several electrical Strong PUF designs against modeling attacks. To that end, we applied various machine learning techniques to challenge-response sets from two sources: (i) Pseudo-random numeric simulations which used an additive delay model, with and without artificially injected errors; and (ii) Silicon CRP data from FPGAs and ASICs. The examined Strong PUFs included standard Arbiter PUFs, XOR Arbiter PUFs, Lightweight Secure PUFs, and Feed-Forward Arbiter PUFs. We also investigated the hardness of certain Ring Oscillator (RO) PUF architectures [46] if used in typical Strong PUF scenarios, i.e., under the presumption that their CRP-interface is publicly accessible. If nothing else, this gives us an indication for how many runs these PUFs can be used securely within (limited count) identification protocols à la Pappu et al. [33], [34]. Some of our main results are summarized in Table XI.

We found that all examined Strong PUF candidates *under a certain size and architectural complexity* could be machine learned with prediction rates above 99%. These rates sometimes are above the practical silicon stability of the examined PUFs. As explained in detail Section II-G, this is not paradoxical, but a natural consequence of our evaluation methology. For example, in silicon attacks an adversary can put the PUFs to stable ambient conditions and apply majority voting to get extremely stable CRP sets.

The attacks required a number of CRPs that grows only linearly or log-linearly in the internal parameters of the PUFs, such as their number of stages, XORs, feed-forward loops, or ring oscillators. Apart from XOR Arbiter PUFs and Lightweight PUFs (whose training times grew quasi-exponentially in their number of XORs for large bitlengths k and small to medium number of XORs l), the training times of the applied machine learning algorithms are low-degree polynomial, too.

We also executed a very detailed proof of concept for silicon CRPs for the two most well-studied and commercially most relevant [9], [10] electrical Strong PUF designs, Arbiter PUFs and XOR Arbiter PUFs. In this process, more than four million CRPs collected from ASICs and FPGAs were used. The similarity of our results on similated and silicon data settles a conjecture that had been posed in earlier versions of this work [41]. It shows that the linear delay model is close

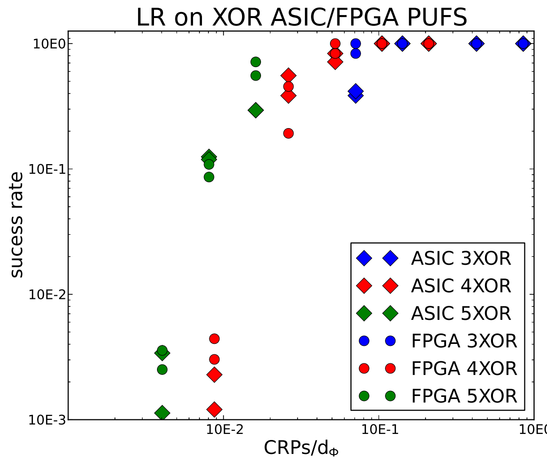

Fig. 12. Average rate of success of the LR algorithm on XOR Arbiter PUFs for FPGA and ASIC data, plotted in dependence of the ratio d_Φ (see Eqn. (11)) to N_{CRP}.

PUF-Type	No. of XORs/ FF-Loops/Ring Osc.	ML Method	Bit Length	CRP Source	CRPs ($\times 10^3$)	Training Time	Prediction Rate
Arbiter PUF	—	LR	128	Simulation	39.2	2.10 sec	99.9%
			64	FPGA	6.5	0.83 sec	99%
			64	ASIC	6.5	0.76 sec	99%
XOR Arbiter PUF	5	LR	128	Simulation	500	16:36 hrs	99%
			64	FPGA	78	39 min	99%
			64	ASIC	78	18:09 min	99%
Lightweight PUF	5	LR	128	Simulation	1000	267 days	99%
FF Arbiter PUF	8	ES	128	Simulation	50	3:15 hrs	99%
Ring Oscillator PUF à la [46] (if used as Strong PUF)	1024	QS	—	Simulation	83.9	—	99%

TABLE XI
SOME OF OUR MAIN RESULTS FOR SIMULATED, NOISE-FREE CRPS AND FOR SILICON CRPS FROM FPGAS AND ASICS. THE PREDICTION RATES AND TRAINING TIMES ARE AVERAGED OVER SEVERAL INSTANCES. ALL PRESENTED TRAINING TIMES ARE CALCULATED AS IF THE ML EXPERIMENT WAS RUN ON ONLY *one single* CORE OF *one single* PROCESSOR. USING k CORES WILL APPROXIMATELY REDUCE THEM BY $1/k$.

to practice, and establishes its use in future security analyses of any Arbiter PUF variants.

Our findings prohibit the use of the modeled architectures *up to a certain size and complexity* in typical Strong PUF protocols whose security rests on the unpredictability or physical unclonability of the Strong PUF, and where the adversary can collect many CRPs via access to the Strong PUF's interface or by eavesdropping protocols. Under the assumption that digital signals can be probed, our results also affect the applicability of the examined Strong PUFs as building blocks in Controlled PUFs, again up to a certain size and complexity. The security of Weak PUFs is not strongly affected by our methods. As discussed in detail in Section I-B, our attacks apply to this PUF type only under the rare circumstance that a Strong PUF is employed inside a hardware system as the Weak PUF, using only few of the many possible challenges of this Strong PUF. Most typical Weak PUFs, such as the SRAM PUF [18], Butterfly PUF [22] or Coating PUF [47], remain unaffected by our attacks.

B. Discussion

Two straightforward, but biased interpretations of our results would be the following: (i) All Strong PUFs are insecure. (ii) The long-term security of electrical Strong PUFs can be restored trivially, for example by increasing the PUF's size. Both views are simplistic, and the truth is more involved.

Starting with (i), our current attacks are indeed sufficient to break several delay-based PUF implementations. But there are a number of ways how PUF designers can fight back in future designs. First, increasing the bitlength k in an XOR Arbiter PUF or Lightweight Secure PUF with l XORs increases the effort of the presented attacks methods as a polynomial function of k with exponent l (in approximation for large k and small or medium l). At the same time, it does not worsen the PUF's stability [9]. For now, one could therefore disable attacks through choosing a strongly increased value of k and a value of l that corresponds to the stability limit of such a construction. For example, an XOR Arbiter PUF with 8 XORs and bitlength of 512 is implementable by standard fabrication processes [9], but is currently beyond the reach of our attacks. Similar considerations hold for Lightweight PUFs of these sizes. Secondly, new design elements may raise the attacker's complexity further, for example adding nonlinearity (such as AND and OR gates that correspond to MAX and MIN operators [24]). Combinations of Feed-Forward and XOR architectures could be hard to machine learn too, partly because they seem susceptible only to different and mutually-exclusive ML techniques.

Moving away from delay-based PUFs, the exploitation of the dynamic characteristics of current and voltage seems promising, for example in analog circuits [7]. Also special PUFs with a very high information content (so-called SHIC PUFs [39], [40], [21]) could be an option, but only in such applications where their slow read-out speed and their comparatively large area consumption are no too strong drawbacks. Their promise is that they are naturally immune against modeling attacks, since all of their CRPs are information-theoretically independent. Finally, optical Strong PUFs, for example systems based on light scattering and interference phenomena [34], show strong potential in creating high input-output complexity.

Regarding view (ii), PUFs are different from classical cryptoschemes like RSA in the sense that increasing their size often likewise decreases their input-output stability. For example, raising the number of XORs in an XOR Arbiter PUF and Lightweight PUF has an exponentially strong effect *both* on the attacker's complexity *and* on the instability of the PUF. We are yet unable to find parameters that increase the attacker's effort *exponentially* while affecting the PUF's stability merely *polynomially*. Nevertheless, one practically viable possibility is to increase the bitlength of XOR Arbiter PUFs and Lightweight PUFs, as discussed above. Future work will have to show whether the described large polynomial growth of the latter method can persist in the long term, or whether its high degree can be diminished by further analysis.

C. Future Work

The upcoming years will presumably witness strong competition between codemakers and codebreakers in the area of Strong PUFs. Similar to the design of classical cryptoprimitives, for example stream ciphers, this process can be expected to converge at some point to solutions that are resilient against the known attacks. Some first attempts into this direction have already been made in [49], [30], [5], [6], but we did not analyze their viability in detail in this work.

For PUF designers, it may be interesting to investigate some of the concepts that we mentioned above. For PUF breakers, a worthwhile starting point is to improve the attacks presented in this paper through optimized implementations and new ML methods. A performance comparison between our results and earlier approaches that used SVMs and comparable techniques [24], [31], illustrates the strong effect of the choice of the right ML-algorithm (see Section I-C). Another, qualitatively new path is to combine modeling attacks with information obtained from direct physical PUF measurements or from side channels. For example, applying the same challenge multiple times gives an indication of the noise level of a response bit. It enables conclusions about the absolute value of the final runtime difference in the PUF. Such side channel information can conceivably improve the success and convergence rates of ML methods, though we have not exploited this in this paper.

Acknowledgements

This work was partially supported by the Physical Cryptography Project of the Technische Universität München, the Semiconductor Research Corporation under Task 1836.074, and the US NSF under grants CNS 0923313 and 0964641.

References

[1] T. Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.

[2] C.M. Bishop et al. *Pattern recognition and machine learning*. Springer New York:, 2006.

[3] A. Blumer, A. Ehrenfeucht, D. Haussler, M.K. Warmuth. Learnability and the Vapnik–Chervonenkis dimension. *Journal of the ACM, 36(4):929–865*, 1989

[4] C. Brzuska, M. Fischlin, H. Schröder, S. Katzenbeisser. Physical Unclonable Functions in the Universal Composition Framework. *CRYPTO 2011.*

[5] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, U. Rührmair. The Bistable Ring PUF: A New Architecture for Strong Physical Unclonable Functions. *HOST 2011.*

[6] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, U. Rührmair. Characterization of the Bistable Ring PUF. *DATE 2012.*

[7] G. Csaba, X. Ju, Z. Ma, Q. Chen, W. Porod, J. Schmidhuber, U. Schlichtmann, P. Lugli, and U. Rührmair. Application of mismatched cellular nonlinear networks for physical cryptography. *IEEE CNNA*, 2010.

[8] I. Damgard, A. Scafuro: Unconditionally Secure and Universally Composable Commitments from Physical Assumptions. *Cryptology ePrint Archive, 2013:108*, 2013.

[9] S. Devadas. Physical unclonable functions and secure processors. *Invited Talk, CHES 2009.*

[10] S. Devadas et al.: Design and Implementation of PUF-Based 'Unclonable' RFID ICs for Anti-Counterfeiting and Security Applications. *IEEE Int. Conf. on RFID*, 2008.

[11] M. van Dijk: *System and method of reliable forward secret key sharing with physical random functions.* US Patent No. 7,653,197, October 2004.

[12] M. van Dijk, U. Rührmair: Physical Unclonable Functions in Cryptographic Protocols: Security Proofs and Impossibility Results. *Cryptology ePrint Archive, 2012:228*, 2012.

[13] Y. Dodis, R. Ostrovsky, L. Reyzin, L., A. Smith: *Fuzzy extractors: How to generate strong keys from biometrics and other noisy data.* SIAM Journal on Computing, 38(1), 97-139, 2008.

[14] B. L. P. Gassend. *Physical random functions*. MSc thesis, MIT, 2003.

[15] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas. Silicon physical random functions. *ACM CCS 2002.*

[16] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Controlled physical random functions. *ACSAC 2002.*

[17] B. Gassend, D. Lim, D. Clarke, M. Van Dijk, and S. Devadas. Identification and authentication of integrated circuits. *Concurrency and Computation: Practice & Experience*, 16(11):1077–1098, 2004.

[18] J. Guajardo, S. Kumar, G.J. Schrijen, and P. Tuyls. FPGA intrinsic PUFs and their use for IP protection. *CHES 2007.*

[19] C. Helfmeier, C. Boit, D. Nedospasov, J.P. Seifert: Cloning Physically Unclonable Functions. *HOST 2013.*

[20] D.E. Holcomb, W.P. Burleson, and K. Fu. Initial sram state as a fingerprint and source of true random numbers for RFID tags. *Conference on RFID Security*, 2007.

[21] C. Jaeger, M. Algasinger, U. Rührmair, G. Csaba, and M. Stutzmann. Random p-n-junctions for physical cryptography. *Applied Physics Letters*, 96(172103), 2010.

[22] S.S. Kumar, J. Guajardo, R. Maes, G.J. Schrijen, and P. Tuyls. Extended abstract: The butterfly PUF protecting IP on every FPGA. *HOST 2008.*

[23] J.W. Lee, D. Lim, B. Gassend, G.E. Suh, M. Van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. *IEEE VLSI Circuits Symposium*, 2004.

[24] D. Lim. *Extracting Secret Keys from Integrated Circuits*. Msc thesis, MIT, 2004.

[25] D. Lim, J.W. Lee, B. Gassend, G.E. Suh, M. Van Dijk, and S. Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration Systems*, 13(10):1200, 2005.

[26] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Testing techniques for hardware security. In *International Test Conference (ITC)*, 2008.

[27] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Lightweight secure pufs. *IEEE/ACM Int. Conf. on Computer-Aided Design*, 2008.

[28] M. Majzoobi, F. Koushanfar and M. Potkonjak. Techniques for Design and Implementation of Secure Reconfigurable PUFs. *ACM Trans. Reconfigurable Technology and Systems*, vol. 2, no.1, 2009.

[29] M. Majzoobi, F. Koushanfar and S. Devadas. FPGA PUF using programmable delay lines. *IEEE Workshop Information Forensics and Security (WIFS)*, 2010.

[30] M. Majzoobi, M. Rostami, F. Koushanfar, D.S. Wallach, and S. Devadas: Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching. *IEEE S&P Workshops*, 2012.

[31] Erdinç Öztürk, Ghaith Hammouri, and Berk Sunar. Towards robust low cost authentication for pervasive devices. *IEEE PerCom*, 2008.

[32] C.H. Papadimitriou. *Computational complexity*. John Wiley, 2003.

[33] R. Pappu. *Physical One-Way Functions*. Phd thesis, MIT, 2001.

[34] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026, 2002.

[35] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *IEEE international conference on neural networks*, 1993.

[36] U. Rührmair. Oblivious transfer based on physical unclonable functions (extended abstract). *TRUST 2010.* LNCS Vol. 6101, Springer, 2010.

[37] U. Rührmair, H. Busch, S. Katzenbeisser: Strong PUFs: Models, Constructions and Security Proofs. In A.-R. Sadeghi, P. Tuyls (Editors): *Towards Hardware Intrinsic Security: Foundation and Practice.* Springer, 2010.

[38] U. Rührmair, S. Devadas, F. Koushanfar: Security based on Physical Unclonability and Disorder. In M. Tehranipoor and C. Wang (Editors): Introduction to Hardware Security and Trust. Springer, 2011.

[39] U. Rührmair, C. Jaeger, M. Bator, M. Stutzmann, P. Lugli, G. Csaba. Applications of high-capacity crossbar memories in cryptography. *IEEE Transactions on Nanotechnology*, 2011.

[40] U. Rührmair, C. Jaeger, C. Hilgers, M. Algasinger, G. Csaba, M. Stutzmann. Security applications of diodes with unique current-voltage characteristics. Financial Cryptography and Data Security (FC), 2010.

[41] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, J. Schmidhuber. Modeling Attacks on Physical Unclonable Functions. *ACM CCS 2010.*

[42] U. Rührmair, J. Sölter, F. Sehnke. On the Foundations of Physical Unclonable Functions. Cryptology ePrint Archive, 2009:277, 2009.

[43] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 1:999–1000, 2010.

[44] H.P.P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc. New York, NY, USA, 1993.

[45] J. Sölter. *Cryptanalysis of Electrical PUFs via Machine Learning Algorithms*. MSc thesis, Technische Universität München, 2009.

[46] G.E. Suh, S. Devadas. Physical unclonable functions for device authentication and secret key generation. *DAC 2007.*

[47] P. Tuyls, G. J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, R. Wolters. Read-proof hardware from protective coatings. *CHES 2006.*

[48] P. Tuyls, B. Skoric. Strong Authentication with PUFs. *In: Security, Privacy and Trust in Modern Data Management, M. Petkovic, W. Jonker (Eds.)*, Springer, 2007.

[49] M.-D. Yu, D. M'Raïhi, R. Sowell, S. Devadas: Lightweight and Secure PUF Key Storage Using Limits of Machine Learning. *CHES 2011.*