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, and Srinivas Devadas.

***Abstract*—We show in this paper how several proposed Strong Physical Unclonable Functions (PUFs) can be broken by numerical modeling attacks. Given a set of challenge-response pairs (CRPs) of a Strong PUF, our attacks construct a computer algorithm which behaves indistinguishably from the original PUF on almost all CRPs. This algorithm can subsequently impersonate the PUF, and can be cloned and distributed arbitrarily. This breaks the security of almost all applications and protocols that are based on the respective PUF.**

The PUFs we attacked successfully include standard Arbiter PUFs and Ring Oscillator PUFs of arbitrary sizes, and XOR Arbiter PUFs, Lightweight Secure PUFs, and Feed-Forward Arbiter PUFs of up to a given size and complexity. The attacks are based upon various machine learning techniques, including a specially tailored variant of Logistic Regression and Evolution Strategies.

Our results were obtained on a large number of CRPs coming from numerical simulations, as well as four million CRPs collected from FPGAs and ASICs. 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 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. They are an accesseible target for adversaries, which raises 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 S 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. Assuming the stability of the PUF's responses, any PUF S hence implements an individual function F_S that maps challenges C_i to responses R_{C_i} of the PUF.

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 [29], [8], key exchange or digital rights management purposes [9].

B. Strong PUFs, Controlled PUFs, and Weak PUFs

There are several subtypes of PUFs, each with its own applications and security features. Three major types, which must explicitly be distinguished in this paper, are *Strong PUFs* [29], [8] [1], *Controlled PUFs* [9], and *Weak PUFs* [11], [12], also called *Physically Obfuscated Keys (POKs)* [7]. [2]

1) Strong PUFs: Strong PUFs are disordered physical systems with a complex challenge-response behavior and very many possible challenges. Their security features are: (i) It must be impossible to physically clone a Strong PUF, i.e., to fabricate a second system which behaves indistinguishably from the original PUF in its challenge-response behavior. This restriction shall hold even for the original manufacturer of the PUF. (ii) A complete determination/measurement of all

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 also been referred to as Physical Random Functions [7], or Physical One-Way Functions [28].

[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 [11], and which has later been discussed by Rührmair et al. in [37], [32], [36], [33].

challenge-response pairs (CRPs) within a limited time frame (such as several days or even weeks) must be impossible, even if one can challenge the PUF freely and has unrestricted access to its responses. This property is usually met by the large number of possible challenges and the finite read-out speed of a Strong PUF. (iii) It must be difficult to numerically predict the response R_C of a Strong PUF to a randomly selected challenge C, even if many other CRPs are known.

Possible applications of Strong PUFs cover key establishment [29], [43], identification [29], and authentication [8]. They also include oblivious transfer [31] and any protocols derived from it, including zero-knowledge proofs, bit commitment, and secure multi-party computation [31]. In said applications, Strong PUFs can achieve secure protocols without the usual, standard computational assumptions concerning the factoring or discrete logarithm problem (albeit their security rests on other, independent computational and physical assumptions). Currently known electrical, circuit-based candidates for Strong PUFs are described in [41], [21], [10], [17], [19]. The formal foundations of Strong PUFs have been discussed in [37], [32].

2) Controlled PUFs: A Controlled PUF as described in [9] uses a Strong PUF as a building block, but adds control logic that surrounds the PUF. The logic prevents challenges from being applied freely to the PUF, and hinders direct read-out of its responses. This logic can be used to thwart modeling attacks. However, if the outputs of the embedded Strong PUF can be directly probed, then it may be possible to model the Strong PUF and break the Controlled PUF protocol.

3) Weak PUFs: Weak PUFs, also called Physically Obfuscated Keys (POKs), may have very few challenges — in the extreme case just one, fixed challenge. Their response(s) R_{C_i} are used to derive a standard secret key, which is subsequently processed by the embedding system in the usual fashion, e.g., as a secret input for some cryptoscheme. Contrary to Strong PUFs, the responses of a Weak are never meant to be given directly to the outside world.

Weak PUFs essentially are a special form of non-volatile key storage. Their advantage is that they may be harder to read out invasively than non-volatile memory like EEPROM. Typical examples include the SRAM PUF [12], [11], Butterfly PUF [16] and Coating PUF [42]. Integrated Strong PUFs have been suggested to build Weak PUFs or Physically Obfuscated Keys (POKs), in which case only a small subset of all possible challenges is used [7], [41].

One important aspect of Weak PUFs is error correction and stability. Since their responses are processed internally as a secret key, error correction must be carried out on-chip and with perfect precision. This often requires the storage of error-correcting helper data in non-volatile memory on (or near) the chip. Strong PUFs usually allow error correction schemes that are carried out by the external recipients of their responses.

C. Modeling Attacks on PUFs

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, and 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. If successful, this breaks the security of the PUF and of any protocols built on it. It is known from earlier work that machine learning (ML) techniques are a natural and powerful tool for such modeling attacks [7], [18], [26], [20], [40]. How the required CRPs can be collected depends on the type of PUF under attack.

1) Strong PUFs: Strong PUFs usually have no protection mechanisms that restrict Eve in challenging them or in reading out their responses. Their responses are freely accessible from the outside, and are usually not post-processed on chip [29], [41], [21], [10], [17], [19]. Most electrical Strong PUFs further operate at frequencies of a few MHz [17]. Therefore even short physical access periods enable the read-out of 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 [29]. Eavesdropping on responses, as well as physical access to the PUF that allows the adversary to apply arbitrary challenges and read out their responses, is part of the established attack model for Strong PUFs.

2) Controlled PUFs: For any adversary that is restricted to non-invasive CRP measurement, modeling attacks can be successfully disabled if one uses a secure one-way hash over the outputs of the PUF to create a Controlled PUF. We note that this requires error correction of the PUF outputs which are inherently noisy [9]. 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. Physical access to the PUF is part of the natural attack model on PUFs, as mentioned above.

3) Weak PUFs: Weak PUFs (or POKs) are only susceptible to model building attacks if a Strong PUF, embedded in some hardware system, is used to implement the Weak PUF. This method has been suggested in [7], [41]. In this case, the internal digital response signals of the Strong PUF to injected challenges have to be probed.

We stress that purely numerical modeling attacks, as presented in this paper, are not relevant for *POKs or Weak PUFs with just one challenge* (such as the Coating PUF, SRAM PUF, or Butterfly PUF). This does not necessarily imply that these PUFs are more secure than Strong PUFs or Controlled PUFs, however. Other attack strategies can be applied, 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. Furthermore, attacking a Controlled PUF via collecting CRPs from the underlying Strong PUF requires substantially more signal probing than breaking a POK/Weak PUF that possesses just one challenge.

D. Our Contributions and Related Work

We describe successful modeling attacks on several known electrical candidates for Strong PUFs, including Arbiter PUFs, XOR Arbiter PUFs, Feed-Forward Arbiter PUFs, Lightweight Secure PUFs, and Ring Oscillator PUFs. The attacks are carried out on simulated CRPs and in part on "real" data collected from FPGAs and ASICs. The prediction rates of our machine learned models on simulated data significantly exceed the known or derived stability of the respective PUFs in silicon in these ranges. The results on silicon data show little performance loss, confirming the viability of our attacks in real-world scenarios. All attacks work for PUFs of up to a given number of inputs (or stages) or complexity.

Our attacks are very feasible on the CRP side. They require an amount of CRPs that grows only linearly or log-linearly in the relevant structural parameters of the attacked PUFs, such as their numbers of stages, XORs, feed-forward loops, or ring oscillators. The computation times needed to derive the models (i.e., to train the employed ML algorithms) are low-degree polynomial, with one exception: The computation times for attacking XOR Arbiter and Lightweight Secure PUFs grow, in approximation for medium number of XORs and large number of stages, super-polynomial in the number of XORs. But the instability of these PUFs also increases exponentially in their number of XORs, whence this parameter cannot be raised at will in practical applications. On the other hand, the number of stages in these two types of PUFs can be increased without significant effect on their instability. This provides a potential lever for making these PUFs more secure without destroying their practicality. Our work thus also points to design requirements by which the security of XOR Arbiter PUFs and Lightweight Secure PUFs against modeling attacks could be upheld in the future.

Our results break the security of Strong PUF-type protocols that are based on one of the broken PUFs. This explicitly includes any identification, authentication, key exchange or digital rights management protocols, such as the ones described in [29], [8], [28], [43], [10]. Under the assumptions and attack scenarios described in Section 1.3, our findings also restrict the use of the broken Strong PUF architectures within Controlled PUFs and as Weak PUFs, if we assume that digital values can be probed.

1) Related Work on Modeling Attacks: This article is an extended journal version of Rührmair et al. [36]. Early work on PUF modeling attacks, such as [10], [18], [26], [20], 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 [36]. 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 [36], the main novelty is that results on a large database of silicon CRPs from ASICs and FPGAs have been added. To this end, ten Arbiter PUF instances on ASICs and ten on FPGAs were utilized for CRP collection. The new result settles an open question from the first version of this work [36], showing that our findings on numerically simulated CRPs carry over with very little performance loss to the silicon case. Majority voting over potentially noisy responses for the same challenge was helpful in breaking silicon Arbiter PUFs for large numbers of XORs, and for achieving optimal prediction rates.

Since the appearance of the first version of this work [36], Hospodar et al. recently published PUF modeling attacks on ASIC data in a piece of interesting work [13]. The main differences to this article are the following. Firstly, we take a more comprehensive perspective on PUFs and modeling attacks, examining five different PUF designs and three different CRP sources, while Hospodar et al. focus on only two designs and one source (ASICs). Secondly, we use larger CRP sets of up to 200,000 silicon CRPs per PUF, while Hospodar et al. use smaller databases of up to 9,000 silicon CRPs per PUF instance. Finally, we employ different ML algorithms that lead to a better performance. Hospodar et al. use Support Vector Machines, SVMs, and Artificial Neural Networks, ANNs. Such techniques had been tested by us in early stages, but were not utilized intensively further on, since other algorithms performed better. To name one example, Hospodar et al. break up to 64-bit, 2-XOR Arbiter PUFs by SVMs and ANNs. These techniques cannot generally attack Feed-Forward Arbiter PUFs (see also Section VI-A). Using different and partly tailor-made algorithms, we were able to break up to 64-bit, 5-XOR Arbiter PUFs, both for FPGA and ASIC data, and to successfully attack the examined Feed-Forward Arbiter PUF architectures on large scales.

2) Entropy Analysis vs. Modeling Attacks: Another useful approach to evaluate PUF security is entropy analysis. Two variants exist: First, to analyze the internal entropy of the PUF. This is similar to the established physical entropy analysis in solid-state systems. A second option is to analyze the statistical entropy of all challenge-response pairs of a PUF; how many of them are independent?

Entropy analysis is a valuable tool for PUF analysis, but it differs from our approach in two aspects. First, it is non-constructive in the sense that it does not tell you how to break a PUF, even if the entropy score is low. Modeling attacks, to the contrary, actually break PUFs. Second, it is not clear if the internal entropy of a circuit-based Strong PUF is a good estimate for its security. The security of a Strong PUF comes from an interplay between its random internal parameters (which can be viewed as its entropy), and its internal model or internal functionality. It is not the internal entropy alone that determines the security. As an example, compare an 8-XOR, 256-bit Arbiter XOR PUF to a standard Arbiter PUF with bitlength of $8 \cdot 256 = 2048$. Both have the same internal entropy, but very different security properties, as we show in the sequel.

E. 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 results for various Strong PUF candidates. They deal with Arbiter PUFs, XOR Arbiter PUFs, Lightweight Arbiter PUFs, Feed-Forward Arbiter PUFs and Ring Oscillator PUFs, in sequence. Section VIII 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 IX.

II. Methodology Section

A. Employed Machine Learning Methods

We evaluated various machine techniques prior to our experiments, including Support Vector Machines (SVMs), Logistic Regression, Evolution Strategies, 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 $p\left(C, t \,|\, \vec{w}\right)$ 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 $p\left(C, t \,|\, \vec{w}\right) = \sigma(tf) = (1 + e^{-tf})^{-1}$. 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:

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

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

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

¿From the different optimization methods which we tested in our ML experiments (standard gradient descent, iterative reweighted least squares, RProp [2] [30]), 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.

In our ML experiments, we used an implementation of LR with RProp programmed in our group, which has been put online, see [14]. 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.

The whole process is similar to training an Artificial Neural Network (ANN) [2]. The model of the PUF resembles the network with the runtime delays resembling the weights of an ANN. Similar to ANNs, we found that RProp makes a very big difference in convergence speed and stability of the LR (several XOR-PUFs were only learnable with RProp). But even with RProp the delay set can end up in a region of the search space where no helpful gradient information is available (local minimum). In such a case we encounter the above described situation of converging on a not sufficiently accurate solution and have to restart the process.

2) Evolution Strategies: Evolution Strategies (ES) [1], [39] 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 [38]. 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, and indicated this also in the caption of the respective tables.

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 can be parallelized in a straightforward manner.

C. PUF Descriptions and Models

1) Arbiter PUFs: Arbiter PUFs (Arb-PUFs) were first introduced in [10] [17] [41]. They consist of a sequence of k stages, for example multiplexers. Two electrical signals race simultaneously and in parallel through these stages. Their exact paths are determined by a sequence of k external bits $b_1 \cdots b_k$ applied to the stages, whereby the i-th bit is applied at the i-th stage. After the last stage, an "arbiter element" consisting of a latch determines whether the upper or lower signal arrived first and correspondingly outputs a zero or a one. The external bits are usually regarded as the challenge C of this PUF, i.e., $C = b_1 \cdots b_k$, and the output of the arbiter element is interpreted as their response R. See [10] [17] [41] for details. The parameter k is often referred to as the bitlength of the Arbiter PUF.

It has become standard to describe the functionality of Arb-PUFs via an additive linear delay model [18] [21] [20]. 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 [18] [21] [20].

In greater detail, the following holds. We denote by $\delta_i^{0/1}$ the runtime delay in stage i for the crossed (1) respectively uncrossed (0) signal path. Then

$$\vec{w} = (w^1, w^2, \ldots, w^k, w^{k+1})^T, \tag{3}$$

where $w^1 = \frac{\delta_1^0 - \delta_1^1}{2}$, $w^i = \frac{\delta_{i-1}^0 + \delta_{i-1}^1 + \delta_i^0 - \delta_i^1}{2}$ for all $i = 2, \ldots, k$, and $w^{k+1} = \frac{\delta_k^0 + \delta_k^1}{2}$.

Furthermore,

$$\vec{\Phi}(\vec{C}) = (\Phi^1(\vec{C}), \ldots, \Phi^k(\vec{C}), 1)^T, \tag{4}$$

where $\Phi^l(\vec{C}) = \prod_{i=l}^k (1 - 2b_i)$ for $l = 1, \ldots, k$.

The output t of an Arb-PUF is 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:

$$t = sgn(\Delta) = sgn(\vec{w}^T \vec{\Phi}). \tag{5}$$

Eqn. 5 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 [41], 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 the XOR Arb-PUF 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 (5) 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:

$$t_{XOR} = \prod_{i=1}^{l} sgn(\vec{w}_i^T \vec{\Phi}_i) = sgn(\prod_{i=1}^{l} \vec{w}_i^T \vec{\Phi}_i) \tag{6}$$

$$= sgn(\underbrace{\bigotimes_{i=1}^{l} \vec{w}_i^T}_{\vec{w}_{XOR}} \underbrace{\bigotimes_{i=1}^{l} \vec{\Phi}_i}_{\vec{\Phi}_{XOR}}) = sgn(\vec{w}_{XOR}^T \vec{\Phi}_{XOR}) \tag{7}$$

Whereas (6) gives a non-linear decision boundary with $l(k+1)$ parameters, (7) 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 [21]. It is similar to the XOR Arb-PUF of the last paragraph. 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, according to the formula

$$o_j = \bigoplus_{i=1,\ldots,x} r_{(j+s+i) \bmod l} \quad \text{for } j = 1, \ldots, m. \tag{8}$$

Thereby the values for m (the number of output bits of the Lightweight PUF), x (the number of values r_j that influence each single output bit) and s (the circular shift in choosing the x values r_j) are variable design parameters.

Another difference to the XOR Arb-PUFs lies in the l inputs $C_1 = b_1^1 \cdots b_k^1, C_2 = b_1^2 \cdots b_k^2, \ldots, C_l = b_1^l \cdots b_k^l$ 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. This input mapping constitutes the most significant difference between the Lightweight PUF and the XOR Arb PUF. We refer the reader to [21] 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 [10] [17] [18] and further discussed in [20]. 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 [17]), 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. We did not consider it our target to develop new FF Arb-PUF architectures with possibly optimized features in this publication.

5) Ring Oscillator PUFs: Ring Oscillator PUFs (RO-PUFs) were discussed in [41], though oscillating loops were proposed in the original silicon PUF paper [8]. They are based on the influence of fabrication variations on the frequency of several, identically designed ring oscillators. While [41] 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 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. In the language of Equ. 3, it amounts to choosing the entries w^i pseudo-randomly. (ii) If a response of this PUF instance to a given challenge is needed, it is calculated by use of the delays selected in step (i), and by application of a linear additive delay model [19]: The delays of the two electrical signal paths are simply added up and compared.

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. Following the PUF block diagram in Figure 1, the Arb-PUFs were composed of 64 pairs of MUXs and a D flip-flop based arbiter, and were implemented in Verilog.

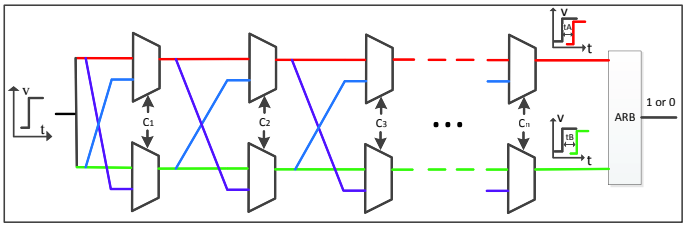

Fig. 1. Arbiter PUF Block Diagram

It is well known from earlier work [25] that in order to implement Arb-PUFs on FPGAs, it is not sufficient to implement a simple functional description. The on-chip routing constraints and the inaccurate arbiter design on FPGAs lead to asymmetric delay skews, which dominate and negate the effect of physical process variations on the path delays. This is indeed the main shortcoming of FPGAs in implementing delay-based PUFs. As a countermeasure, Majzoobi et al. [22] and [23] proposed a lookup table (LUT) based Programmable Delay Line (PDL) (see Figure 2) to tune the delay skew. This PDL realizes pico-second resolution, and moreover, it can be implemented within a single LUT. By applying different LUT input vectors, the input to the PDL will be delayed accordingly to tune the delay skew from asymmetric routing. We followed this strategy in our implementation.

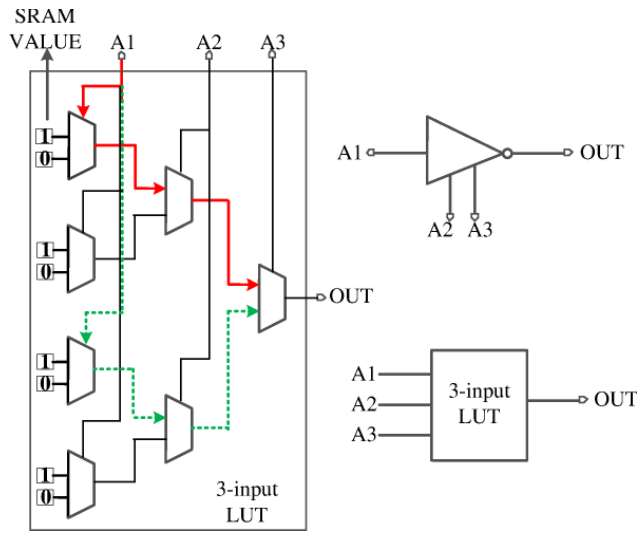

Fig. 2. Programmable Delay Line

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" 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

$$F = 1 + X^1 + X^3 + X^4 + X^{64} \tag{9}$$

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 silicon, we built Arb-PUF circuits with 45nm SOI CMOS ASICs. The Arb-PUF circuit is composed of a set of delay elements and an arbiter circuit element, as illustrated in Figure 1. Each delay element consists of two multiplexers (MUXes) with their inputs connected. The challenge vectors form the select inputs to the MUXs, which determine the paths taken by the top and bottom signal, respectively. To evaluate the response bit for a particular challenge, an input rising edge is propagated through the delay stages. The response bit is determined to be a "1" or "0" based on the top and bottom signal arrival times. In this PUF circuit, 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, see Figure 3. 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.

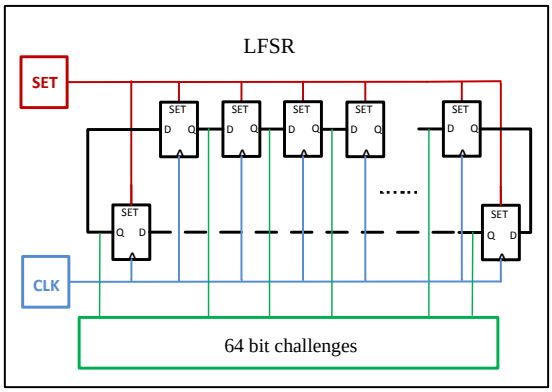

Fig. 3. Challenge Generator of ASIC Arbiter PUFs

40 unpackaged chips of 45nm SOI CMOS technology were taped out for post-silicon measurement, a die photo of which can be seen in Figure 4. 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 shown in Figure 5. 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.

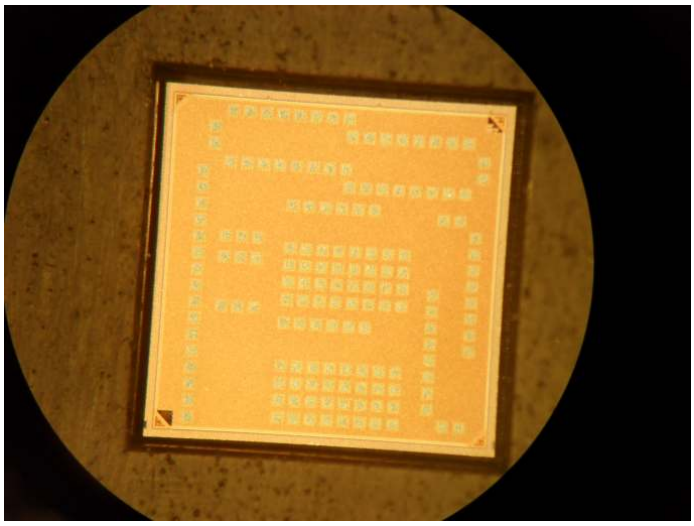

Fig. 4. Die Photo of ASIC Arbiter PUFs

There exists noise in the test-bed and equipment we utilized. 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. For example, if we get more than three "1s" or "0s", response of that challenge is marked as "1" or "0". We captured 200,000 CRPs from each of the ten used PUF instances, resulting in a total of two million CRPs collected from ASICs.

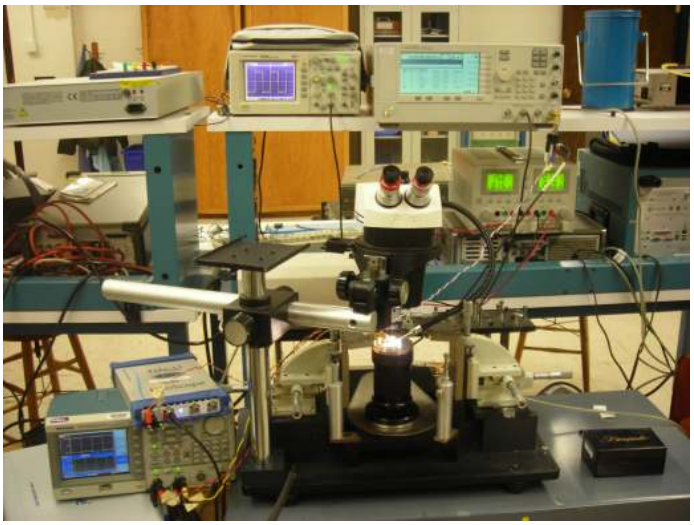

Fig. 5. CRP Collection from ASIC Arbiter PUFs

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).

III. ARBITER PUFS

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 $\pm 2\%$ [3]. 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.

B. Scalability

We also executed scalability experiments with LR, which are displayed in Figure 6 and Figure 7. 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) 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

$$N_{CRP} = O(k/\epsilon). \tag{10}$$

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 6 with a slope of $a = -1$, we obtained the following empirical formula (11). It gives the approximate number of CRPs N_{CRP} that is required to learn a k-stage arbiter PUF with error rate ϵ:

$$N_{CRP} \approx 0.5 \cdot \frac{k+1}{\epsilon} \tag{11}$$

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 7, 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

$$N_{BOP} = O\left(\frac{k^2}{\epsilon} \cdot \log\frac{k}{\epsilon}\right). \tag{12}$$

IV. XOR ARBITER PUFS

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. 7), one can also specify the non-linear boundary (Eqn. 6). This

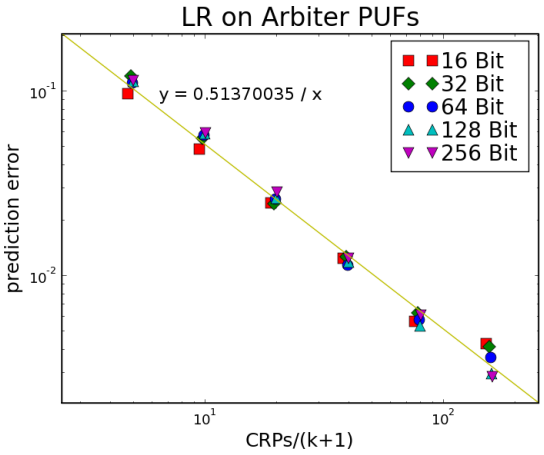

Fig. 6. Double logarithmic plot of misclassification rate ϵ on the ratio of training CRPs N_{CRP} and $dim(\Phi) = k + 1$.

[3]The exact figures reported in [18] are: 4.57% CRP variation for a temperature variation of 45C, and 2.16% for a voltage variation of $\pm 2\%$.

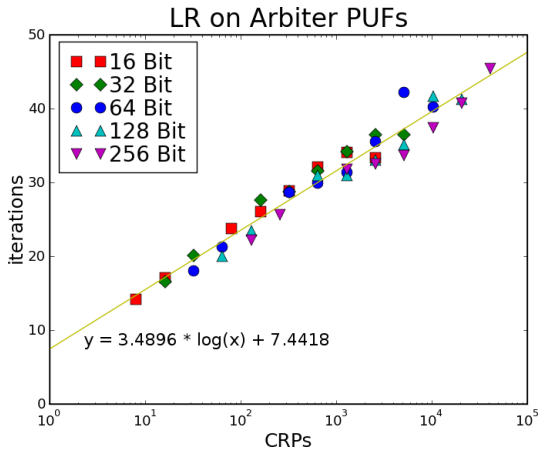

Fig. 7. No. of iterations of the LR algorithm until "convergence" occurs (see section II), plotted in dependence of the training set size N_{CRP}.

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. Performance on Error-Inflicted CRPs

The CRPs used in Section IV-A have been generated pseudorandomly via an additive, linear delay model of the PUF. This deviates from reality in two aspects: First of all, the CRPs obtained from real PUFs are subject to noise and random errors. Secondly, the linear model matches the phenomena on a real circuit very closely [18], but not perfectly. This leads to a deviation of any real system from the linear model on a small percentage of all CRPs.

In order to mimic this situation, we investigated the ML performance when a small error is injected artificially into the training sets. A given percentage of responses in the training set were chosen randomly, and their bit values were flipped. Afterwards, the ML performance on the unaltered, error-free test sets was evaluated. The results are displayed in Tables III and IV. They show that LR can cope very well with errors, provided that around 3 to 4 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.

C. Scalability

Figures 9 and 10 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):

$$N_{CRP} \sim \frac{(k+1) \cdot l}{\epsilon}. \tag{13}$$

But, in contrast to standard Arb-PUFs, optimizing the non-linear decision boundary (6) 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

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. TRAINING TIMES ARE AVERAGED OVER DIFFERENT PUF-INSTANCES.

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 III
LR ON 128-BIT, 4-XOR ARB PUFS WITH DIFFERENT LEVELS OF ERROR IN THE TRAINING SET. 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.

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

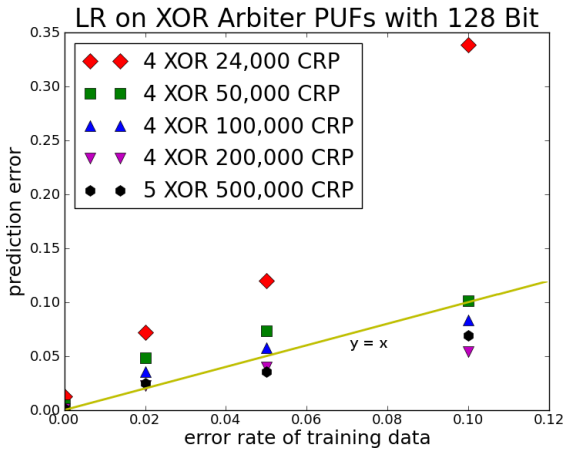

Fig. 8. Graphical illustration of the effect of error on LR in the training set, with chosen data points from Tables III and IV.

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 [40], 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 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

$$\begin{aligned} d_\Phi &= \binom{k+1}{l} + \binom{k+1}{l-2} + \binom{k+1}{l-4} + \ldots \\ &\overset{k \gg l}{\approx} \frac{(k+1)^l}{l!}. \end{aligned} \tag{14}$$

The approximation applies when k is considerably larger than l, which holds for the considered PUFs for stability reasons. Following [40], 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

$$N_{trial} = O\left(\frac{d_\Phi}{N_{CRP}}\right) = O\left(\frac{(k+1)^l}{N_{CRP} \cdot l!}\right). \tag{15}$$

Furthermore, each trial has the complexity

$$T_{trial} = O\left((k+1) \cdot l \cdot N_{CRP}\right). \tag{16}$$

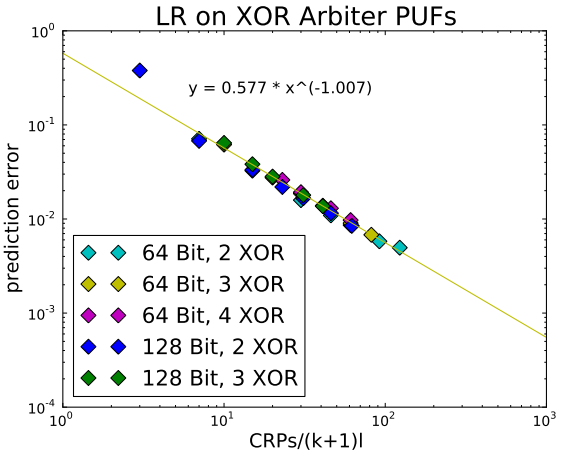

Fig. 9. Double logarithmic plot of misclassification rate ϵ on the ratio of training CRPs N_{CRP} and problem size $dim(\Phi) = (k+1) \cdot l$.

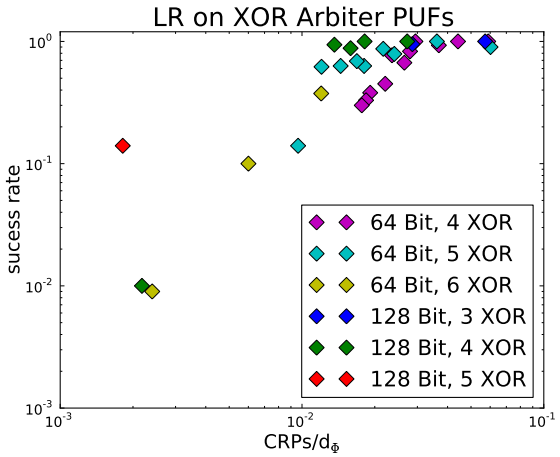

Fig. 10. Average rate of success of the LR algorithm plotted in dependence of the ratio d_Φ (see Eqn. (14)) to N_{CRP}.

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 V
LR ON LIGHTWEIGHT PUFS. PREDICTION RATE REFERS TO SINGLE OUTPUT BITS. TRAINING TIMES WERE AVERAGED OVER DIFFERENT PUF INSTANCES.

V. LIGHTWEIGHT SECURE PUFS

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 V. The specific design of the Lightweight PUF improves its ML resilience by a notable quantitative factor, especially with respect to the training times and CRPs. The given training times and prediction rates relate to single output bits of the Lightweight PUF.

B. Scalability

Some theoretical consideration [40] 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. 13) 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. 15. It seems to lie between $O\big(\frac{(k+1)^l}{N_{CRP}\cdot l!}\big)$ and the related expression $O\big(\frac{(k+1)^l}{N_{CRP}}\big)$ [40]. 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. The specific challenge architecture of the Lightweight PUF hence noticeably complicates the life of an attacker in practice.

VI. Feed Forward Arbiter PUFs

A. Machine Learning Results

We experimented with SVMs and LR on FF Arb-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 [20].

The application of ES finally allowed us to tackle much more complex FF-architectures with up to 8 FF-loops. 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 VI 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 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% [18].

B. Results on Error-Inflicted CRPs

For the same reasons as in Section IV-B, we evaluated the performance on error-inflicted CRPs with respect to ES and FF Arb PUFs. The results are shown in Table VII and Figure 11. ES possesses an extremely high tolerance against the inflicted errors; its performance is hardly changed at all.

C. 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 12. 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 13. Again, each data point shows the averaged prediction error of 10 trials on the same, random PUF instance.

In contrast to the Sections IV-C 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

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 VI
ES on Feed-Forward Arbiter PUFs. 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.

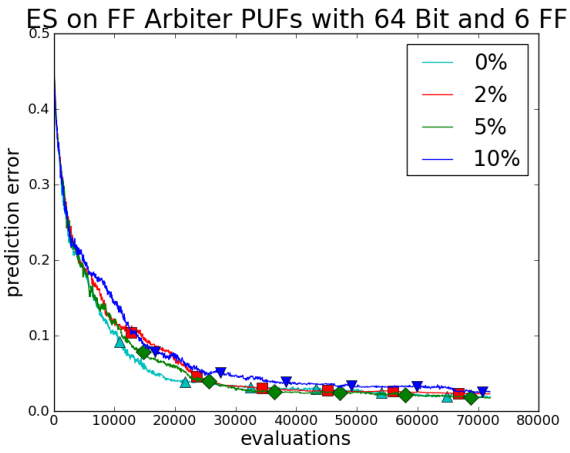

Fig. 11. Graphical illustration of the tolerance of ES to errors. We show the best result of 40 independent trials on one randomly chosen PUF instance for varying error levels in the training set. The results hardly differ.

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 VII
ES on 64-bit, 6 FF Arb PUFs with different levels of error in the training set. 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.

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.

It still seems somewhat suggestive from the data points in Figures. 12 and 13 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

$$N_{CRP} = O(s/\epsilon^c) \tag{17}$$

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

$$N_{CRP} \approx 9 \cdot \frac{s+1}{\epsilon^{3/4}}. \tag{18}$$

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. 18 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

$$N_{BOP} = O(s^3/\epsilon^c) \tag{19}$$

for some constant $0 < c < 1$.

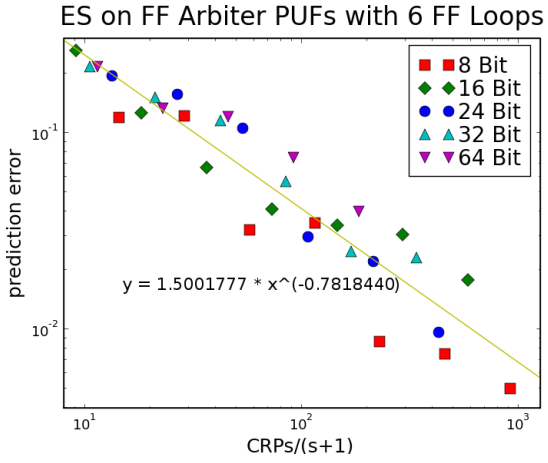

Fig. 12. Results of 10 trials per data point with ES for different lengths of FF Arbiter PUFs and the hyperbola fit.

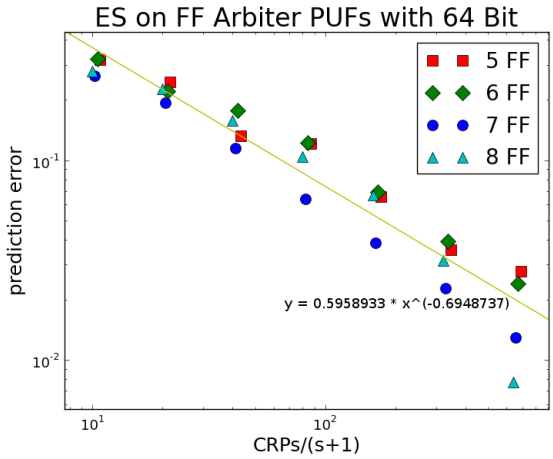

Fig. 13. Results of 10 trials per data point with ES for different numbers of FF-loops and the hyperbola fit.

VII. Ring Oscillator PUFs

A. Possible Attacks

There are several strategies to attack a RO-PUF. The most straightforward attempt is a simple read out of all CRPs. This is easy, since there are just $k(k-1)/2 = O(k^2)$ CRPs of interest, given k ring oscillators.

If Eve is able to choose the CRPs adaptively, she can employ a standard sorting algorithm to sort the RO-PUF's frequencies $(f_1, \ldots, f_k)$ in ascending order. This strategy subsequently allows her to predict all outputs with 100% correctness, without knowing the exact frequencies f_i themselves. The time and CRP complexities of the respective sorting algorithms are well known [27]; 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 VIII
Quick Sort applied to the Ring Oscillator PUF. The given CRPs are averaged over 40 trials.

The most interesting case for our investigations is when Eve cannot adaptively choose the CRPs she obtains, but still wants to achieve optimal prediction rates. This case occurs in practice whenever Eve obtains her CRPs from protocol eavesdropping, for example. We carried out experiments for this case, in which we applied Quick Sort (QS) to randomly drawn CRPs.

The results are shown in Table VIII. The estimated required number of CRPs is given by

$$N_{CRP} \approx \frac{k(k-1)(1-2\epsilon)}{2+\epsilon(k-1)}, \tag{20}$$

and the training times are low-degree polynomial. Eqn. 20 quantifies the limited-count authentication capabilities of RO-PUFs.

VIII. Proof of Concept for Silicon Data

So far, all of our results were achieved on numerically simulated CRPs. In any simulations of the Arbiter PUF and its variants, an additive linear delay model has been used (see Section II-C). It assumes that the runtime delays in each subcomponent simply add up linearly to form an overall delay of each signal path. Based on earlier experiments with silicon implementations [18] [6], 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 [36].

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 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).

Below, we evaluate the absolute performance and also the scalability on this CRP data. Our findings confirm that there is little perfomance loss for silicon CRPs. Among other things, this establishes the good applicability of the linear additive delay model in any future security analyses of delay-based PUFs.

A. Results on 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. Table IX gives the results of 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 difficult to achieve extremely low prediction rates such as 0.1% or smaller; a very large amount of CRP data would be required for this rate. Anyway, in practice a prediction error of 1% or below is already sufficient to break the system.

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.

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 for silicon CRP data from FPGAs.

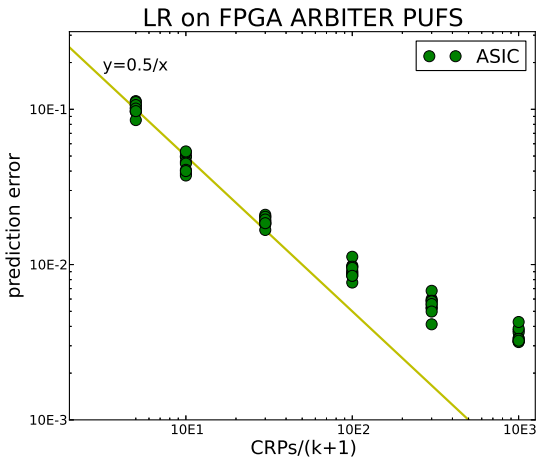

Fig. 14. Performance of LR on FPGA Arbiter PUFs for small prediction errors. Each data point represents a single PUF instance.

Figure 14 depicts the results of our scaling experiments on the required number of CRPs for FPGA data, while Figure 15 shows the same for ASIC data. The figures show 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:

$$N_{CRP} \approx 0.5 \cdot \frac{k+1}{\epsilon}. \tag{21}$$

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.

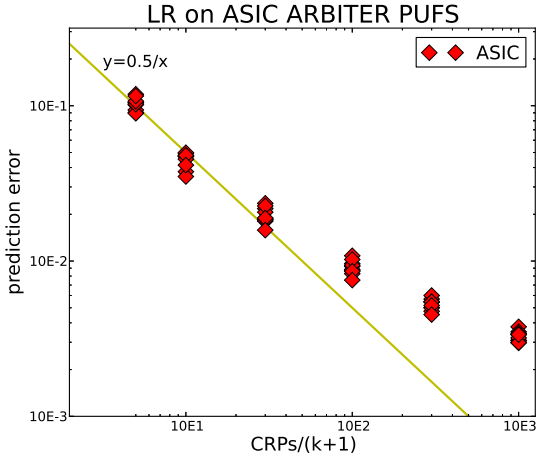

Fig. 15. Performance of LR on ASIC Arbiter PUFs for small prediction errors. Each data point represents a single PUF instance.

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 [22], [23] 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 16. Our results can be seen as confirmation that the basic relationship given in Section IV-C still holds, and that the runtime scales as

$$N_{BOP} = O\left(\frac{k^2}{\epsilon} \cdot \log \frac{k}{\epsilon}\right). \tag{22}$$

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.

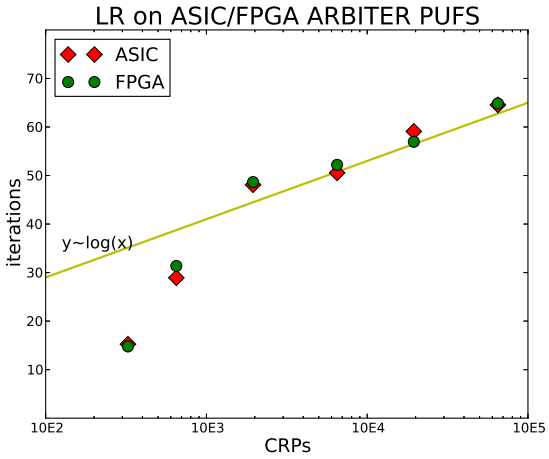

Fig. 16. Necessary trials for LR on FPGA and ASIC Arbiter PUFs.

B. Results on 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-C. The required number of CRPs vs. the achieved prediction error is shown in Figure 17. 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

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 ARBITER PUFS, ALL OF BITLENGTH 64, FOR FPGA AND ASIC DATA. TRAINING TIMES ARE AVERAGED OVER DIFFERENT PUF-INSTANCES.

rates, and slowly increases with the number of XORs. Still, the saturation is so mild that also prediction errors below 1% can be achieved, provided that a sufficient amount of CRPs is used. Over 1%, the basic relationship

$$N_{CRP} \sim \frac{(k+1) \cdot l}{\epsilon}. \tag{23}$$

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

In terms of computation times, our findings are summarized in Figure 18. It corresponds to Figure 10 in Section IV-C, 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

$$N_{trial} = O\left(\frac{d_\Phi}{N_{CRP}}\right) = O\left(\frac{(k+1)^l}{N_{CRP} \cdot l!}\right). \tag{24}$$

Furthermore, each trial again has the approximate complexity

$$T_{trial} = O\left((k+1) \cdot l \cdot N_{CRP}\right). \tag{25}$$

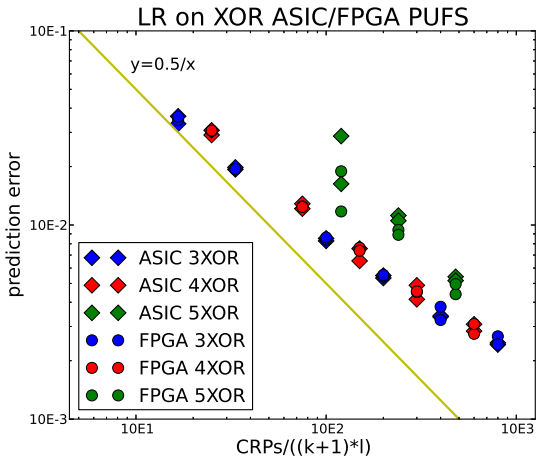

Fig. 17. Performance of LR on XOR Arbiter PUFs for FPGA and ASIC data, for small prediction errors.

IX. SUMMARY AND DISCUSSION

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; and (ii) Silicon CRP data from FPGAs and ASICs. Some of our main results are summarized in Table XI.

We found that all examined Strong PUF candidates under a given size could be machine learned with prediction rates above 99 %. 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.

The majority of our results were obtained on numerically simulated CRPs. In order to show their viability for silicon systems, we executed a very detailed proof of concept for the two most well-studied 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 [36]. It shows that the linear delay model is close to practice, and establishes its use in future security analyses of any Arbiter PUF variants.

Our findings prohibit the use of the broken architectures as Strong PUFs or in Strong-PUF based protocols. Under the assumption that digital signals can be probed, they also affect the applicability of the cryptanalyzed PUFs as building blocks in Controlled PUFs and Weak PUFs.

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.

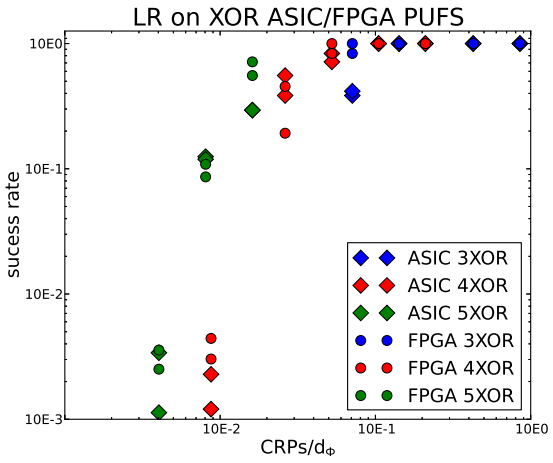

Fig. 18. Average rate of success of the LR algorithm on XOR Arbiter PUFs for FPGA and ASIC data, plotted in dependence of the ratio d_Φ (see Eqn. (14)) to N_{CRP}.

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 [6]. 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 [6], 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 [18]). 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 [5]. Also special PUFs with a very high information content (so-called SHIC PUFs [34], [35], [15]) 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 [29], 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 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, as discussed above. Future work will have to show whether the described large polynomial growth 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 an intense 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

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	6500	0.83 sec	99%
			64	ASIC	6500	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	1024	QS	–	Simulation	83.9	—	99%

TABLE XI

SOME OF OUR MAIN RESULTS. 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 $1/k$.

the known attacks. Some first attempts into this direction have already been made in [44], [24], [3], [4], but we did not analyze their viability 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 [18], [13], illustrates the strong effect of the choice of the right ML-algorithm (see Section I-D). 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] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, U. Rührmair: *The Bistable Ring PUF: A New Architecture for Strong Physical Unclonable Functions*. HOST 2011.

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

[5] 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. In *12th IEEE CNNA - International Workshop on Cellular Nanoscale Networks and their Applications*. Berkeley, CA, USA, February 3 - 5 2010.

[6] S. Devadas. Physical unclonable functions and secure processors. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009)*, September 2009.

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

[8] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, page 160. ACM, 2002.

[9] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Controlled physical random functions. In *Proceedings of 18th Annual Computer Security Applications Conference*, Silver Spring, MD, December 2002.

[10] 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.

[11] J. Guajardo, S. Kumar, G.J. Schrijen, and P. Tuyls. FPGA intrinsic PUFs and their use for IP protection. *Cryptographic Hardware and Embedded Systems-CHES 2007*, pages 63–80, 2007.

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

[13] G. Hospodar, R. Maes, and I. Verbauwhede. Machine Learning Attacks on 65nm Arbiter PUFs: Accurate Modeling poses strict Bounds on Usability. In *4th IEEE International Workshop on Information Forensics and Security (WIFS 2012)*, 2012.

[14] http://www.pcp.in.tum.de/code/lr.zip, 2010.

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

[16] S.S. Kumar, J. Guajardo, R. Maes, G.J. Schrijen, and P. Tuyls. Extended abstract: The butterfly PUF protecting IP on every FPGA. In *IEEE International Workshop on Hardware-Oriented Security and Trust, 2008. HOST 2008*, pages 67–70, 2008.

[17] J.W. Lee, D. Lim, B. Gassend, G.E. Suh, M. Van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *Proceedings of the IEEE VLSI Circuits Symposium*, pages 176–179, 2004.

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

[19] 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.

[20] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Testing techniques for hardware security. In *Proceedings of the International Test Conference (ITC)*, pages 1–10, 2008.

[21] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Lightweight secure pufs. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 670–673. IEEE Press, 2008.

[22] 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.

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

[24] 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 Symposium on Security and Privacy Workshops*, pages 33-44, 2012.

[25] S. Morozov, A. Maiti, P. Schaumont. An analysis of delay based PUF implementations on FPGA. *6th International Symposium on Applied Reconfigurable Computing. LNCS*, vol. 5992 (2010), pp. 382–387.

[26] Erdinç Öztürk, Ghaith Hammouri, and Berk Sunar. Towards robust low

cost authentication for pervasive devices. In *PerCom*, pages 170–178. IEEE Computer Society, 2008.
[27] C.H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.
[28] R. Pappu. *Physical One-Way Functions*. Phd thesis, MIT, 2001.
[29] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026, 2002.
[30] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE international conference on neural networks*, volume 1993, pages 586–591. San Francisco: IEEE, 1993.
[31] U. Rührmair. Oblivious transfer based on physical unclonable functions (extended abstract). *TRUST 2010*, Volume 6101 of *Lecture Notes in Computer Science*, pages 430–440. Springer, 2010.
[32] U. ührmair, 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.
[33] U. ührmair, 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.
[34] U. Rührmair, C. Jaeger, M. Bator, M. Stutzmann, P. Lugli, and G. Csaba. Applications of high-capacity crossbar memories in cryptography. *IEEE Transactions on Nanotechnology*, 2011.
[35] U. Rührmair, C. Jaeger, C. Hilgers, M. Algasinger, G. Csaba, and M. Stutzmann. Security applications of diodes with unique current-voltage characteristics. Financial Cryptography and Data Security (FC 2010). *Lecture Notes in Computer Science*, Vol. 6052, Springer, 2010.
[36] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling Attacks on Physical Unclonable Functions. ACM Conference on Computer and Communications Security (CCS'10), 2010.
[37] U. Rührmair, J. Sölter, F. Sehnke. On the Foundations of Physical Unclonable Functions. IACR Cryptology ePrint Archive 2009: 277 (2009).
[38] 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.
[39] H.P.P. Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc. New York, NY, USA, 1993.
[40] J. Sölter. *Cryptanalysis of Electrical PUFs via Machine Learning Algorithms*. MSc thesis, Technische Universität München, 2009.
[41] G.E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. *Proceedings of the 44th annual Design Automation Conference*, page 14, 2007.
[42] P. Tuyls, G. J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters. Read-proof hardware from protective coatings. *Cryptographic Hardware and Embedded Systems-CHES 2006*, pages 369–383, 2006.
[43] P. Tuyls and B. Skoric. Strong Authentication with PUFs. *In: Security, Privacy and Trust in Modern Data Management, M. Petkovic, W. Jonker (Eds.)*, Springer, 2007.
[44] M.-D. Yu, D. M'Rahi, R. Sowell, and S. Devadas: Lightweight and Secure PUF Key Storage Using Limits of Machine Learning. CHES 2011: 358-373, 2011.