Power and Timing Side Channels for PUFs and their Efficient Exploitation

Ulrich Rührmair, Xiaolin Xu, Jan Sölter, Ahmed Mahmoud, Farinaz Koushanfar, Wayne Burleson

***Abstract*—This paper discusses combined modeling and side channel attacks on Strong Physical Unclonable Functions (Strong PUFs). We illustrate our method by the example of the two currently most secure (CCS 2010, IEEE T-IFS 2013) electrical Strong PUFs, so-called XOR Arbiter PUFs and Lightweight PUFs, and successfully attack them at sizes and complexities far beyond the reach of pure modeling techniques (CCS 2010, IEEE T-IFS 2013).**

Our approach makes use of the first power and timing side channels on PUFs reported in the literature. Both provide information on the single outputs of the many parallel Arbiter PUFs inside an XOR Arbiter PUF or Lightweight PUF, and indicate how many of these single outputs (in sum) were equal to one (and how many were equal to zero) before they entered the final XOR gate. Taken for itself, this side channel information is of little value. But if combined with suitably adapted machine learning techniques, it substantially changes attack performance: It reduces the empirically estimated complexities for modeling the above two PUFs from exponential (CCS 2010, IEEE T-IFS) to low degree polynomial.

The practical viability of our attacks is firstly demonstrated by SPICE simulations, and by subsequent ML experiments on numerically simulated CRPs. We thereby confirm attacks on the two above PUFs for up to 16 XORs and challenge bitlengths of up to 512. Secondly, we execute a full experimental proof-of-concept for our timing side channel, successfully attacking FPGA-implementations of the two above PUF types for 8, 12, and 16 XORs, and bitlengths 64, 128, 256 and 512. We implement these sizes for the first time in the literature in silicon, and subsequently attack them successfully by our new methods. We remark that in recent works (CCS 2010, IEEE T-IFS 2013), 8 XOR architectures with bitlength 512 had been explicitly suggested as secure and beyond the reach of current attacks.

Finally, we discuss efficient countermeasures against our power and timing side channels. They could and should be used to secure future Arbiter PUF generations against the latter.

***Keywords*-Physical unclonable functions (PUFs), side channel attacks, power side channel, timing side channel, modeling attacks, machine learning, hardware security**

I. Introduction

Most modern cryptographic and security schemes are built on the concept of a secret key. This forces current hardware to contain a piece of digital information that is, and remains, unknown to the adversary. This requirement can be difficult to uphold in practice: Physical attacks like invasive, semi-invasive or side-channel attacks, as well as software attacks like malware, can lead to key exposure and full security breaks.

Indeed, one of the main motivations in the development of *Physical Unclonable Functions (PUFs)* was their promise to better protect secret digital keys in vulnerable hardware systems. A PUF is an (at least partly) disordered physical system P that can be excited with external stimuli or so-called challenges C_i. It reacts with corresponding responses R_i, which depend on the challenge and on the micro- or nanoscale structural disorder that is present in the PUF. It is assumed that this disorder cannot be cloned or reproduced exactly, not even by the PUF's original manufacturer, and that it is unique to each PUF. Assuming efficient error correction on the noisy PUF responses, each PUF P thus implements a unique and individual function f_P that maps challenges C_i from an admissible challenge set to responses $R_i = f_P(C_i)$. The tuples (C_i, R_i) are thereby usually called the challenge-response pairs (CRPs) of the PUF.

Due to their complex internal structure, PUFs promise to avoid some of the shortcomings of classical digital keys. It is usually harder to read out, predict, or derive PUF-responses than to obtain digital keys that are stored in non-volatile memory. The PUF-responses are only generated when needed, which means that no secret keys are present permanently in an easily accessible digital form. Furthermore, certain types of PUFs promise some natural tamper sensitivity, as their exact behavior depends on minuscule manufacturing irregularities in different layers of the IC. Removing or penetrating these layers is often assumed to automatically change the PUF's read-out values.

These observations have been exploited in the past for various PUF-based security protocols. Prominent examples include schemes for identification [24], [6] or different forms of (tamper sensitive) key storage and applications thereof, such as intellectual property protection or read-proof memory [8], [14], [36]. Simultaneously, the use of PUFs in advanced cryptographic protocols has been investigated. Several protocols were suggested, including schemes for identification [24], key exchange [24], [5], [2], oblivious transfer [26], [2], or bit commitment [23], [2]. Using Strong PUFs in these protocols has the advantage that no permanently stored digital secret keys and standard computational assumptions (such as the hardness of factoring) are involved. Overall, the assumed security advantages of PUFs have attracted considerable attention within the security community over the last decade.

a) Recent Attacks on PUFs and Related Work: In the last years, an increasing number of attacks on PUFs have been published. Some of them were specifically developed for this new primitive, while others are an adaption of known strategies to the PUF case. Not all attacks apply to every PUF design

Ulrich Rührmair, ruehrmair@ilo.de

Xiaolin Xu and Wayne Burleson are with the University of Massachusetts Amherst, Amherst, MA 01003, USA.

Jan Sölter is with the Freie Universität Berlin, 14195 Berlin, Germany.

Ahmed Mahmoud is with the Technische Universität München, 80333 München, Germany.

Farinaz Koushanfar is with Rice University, Houston, TX 77005, USA.

in the same manner. In order to categorize existing work, it makes sense to distinguish between the two major PUF types of *"Weak PUFs"* and *"Strong PUFs"* [1] .

Let us start with *Strong PUFs*. The currently most relevant attack form on this PUF type are machine-learning (ML) based modeling attacks, which have been pursued by a large number of authors [13], [22], [17], [16], [10], [3], [32], [34]. They are particularly well applicable to Strong PUFs, since this PUF type has a publicly accessible CRP interface, which allows the collection of the large numbers of CRPs that are required in these attacks. Two recent works from CCS 2010 and IEEE T-IFS 2013 [32], [34], which represent the current state of the art, have indeed succesfully tackled a considerable number of Strong PUF designs, including standard Arbiter PUFs, XOR Arbiter PUFs, Feed-Forward Arbiter PUFs, Lightweight PUFs, and certain forms of Ring Oscillator PUFs [35], [32], [16]. The architectures with the highest ML-resilience were found to be XOR Arbiter PUFs [35], [32] and Lightweight PUFs [16]. They could only be be attacked efficiently if they are composed of at most five or six single parallel Arbiter PUFs with bitlength 64 or 128 each [32], [34]. To give a full picture to the readers, some concrete results of [32], [34] are summarized in Table I. They show that the CRP consumption and training times grow substantially for larger number of XORs k and bitlengths n. The authors of [32] estimate that the growth is exponential in the number of XORs k, and polynomial to the degree k in the bitlength n. Furthermore, from all PUFs examined in [32], [34], the XOR Arbiter PUF and the Lightweight PUF possess the highest ML resilience, with the latter yet being somewhat harder to break.

To summarize, no efficient attacks on Lightweight PUFs and XOR Arbiter PUFs with bitlengths of 256 or more and with 6 XORs or more were possible prior to this work, and also no polynomial time ML algorithms for attacking these structures were known. Note here that the instability of XOR Arbiter PUFs and Lightweight PUFs with k XORs grows exponentially in k. This puts a natural limit on the number of XORs that can be used. The authors of [32], [34] hence explicitly suggested the use of XOR Arbiter PUFs and Lightweight PUFs with eight parallel Arbiter PUFs (with "8 XORs") and bitlengths 512 as both stable and secure against existing modeling attacks.

Another attack on Strong PUFs that is relevant in our context has been presented recently by Delvaux and Verbauwhede [3]. They combine a noise-based side channel with analytical modeling techniques in order to attack standard Arbiter PUFs (i.e., without any XORs). Interestingly, their modeling approach does not employ any machine learning algorithms. Their method is very attractive, but in its current form achieves slightly worse accuracy than ML-based modeling *without* side channels (97% in [3] vs. 99% in [34]). It also has a higher CRP-consumption than pure ML-based modeling without side channels. This is in opposition to the attacks presented in this paper, actually drastically increase the performance of pure ML-based Strong PUF attacks.

Other recent attacks have mostly focused on so-called *Weak PUFs*. Merli et al. attack the error-correcting module of Weak PUFs at TRUST 2011 [19], and EM analyses on ring oscillator PUFs (RO PUFs) have been carried out by the same group at WESS 2011 [20]. Nedospasov et al. describe successful invasive methods on SRAM PUFs at FDTC 2013 [21]. Finally, Helfmeier et al. report cloning attacks on SRAM PUFs at HOST 2013 [9].

Comparable cloning or invasive attacks on Strong PUFs have not been reported to this date. There is a reason for this: Typical Strong PUFs possess very many possible challenges and a complex response generation process, in which a large number of components interact to produce one single response. Successful physical cloning would require the accurate duplication of all of these components in order to get all CRPs right. In the case of an Arbiter PUF, for example, it would involve successful tuning of *all* delay values in the subcomponents; or in the case of optical PUFs, it would require precise positioning of *all* the scattering centers in the PUF. This seems significantly harder than tuning the single output of typical Weak PUFs, for example tuning the start-up value of a single SRAM cell. Furthermore, an invasive read-out of the single response of a Weak PUF is obviously simpler than reading out the exponentially many CRPs of typical Strong PUFs.

Finally, a number of protocol attacks on advanced Strong PUF schemes such as key exchange or oblivious transfer have been presented recently at CHES 2012, IEEE S&P 2013, and other venues [31], [27], [28], [29]. They describe relevant methods, but lie beyond the topic of this paper, which focuses on hardware attacks.

b) Our Contributions: This work combines for the first time machine learning based modeling with side channel information to reach new performance levels in Strong PUF attacks. We illustrate our methods by the example of XOR Arbiter PUFs and Lightweight PUFs, which are the two most secure and practical electrical Strong PUFs according to the current state of the art [32], [34]. We show that our new approach allows the first successfully attacks on these two designs for up to 16 internal, parallel single Arbiter PUFs (i.e., for "16 XORs"), and for bitlengths of up to 512. It also reduces, for the very first time, the CRP consumption of the attacks to linear, and the corresponding ML training times to low degree polyonmial.

One part of our attacks is executed on simulated CRPs, which were generated by an additive linear delay model, and under the assumption of a noise-free side channel information. Both assumptions do not strongly diminish the relevance of our results, however: Rührmair et al. [32], [34] showed that for any Arbiter PUF variants, results on simulated CRPs generated by the linear additive delay model carry over with very little performance loss to silicon CRPs. Furthermore, the same works [32], [34] showed that the ML method employed in this paper (i.e., logistic regression) possess substantial error tolerance. Reasonable noise levels in the side channel will not alter its performance significantly.

In addition, we carry out a full proof of concept on FPGA implementations of XOR Arbiter PUFs and Lightweight PUFs. In this process, we utilize an efficient, yet simple error

[1] For an explanation of Weak and Strong PUFs and other PUF types, we refer the reader to Rührmair et al. [32], [34], [30].

PUF-Type	No. of XORs	Bit Length	Source of CRPs	CRPs ($\times 10^3$)	Prediction Rate	Training Time
Arbiter PUF	—	64 128	Simulation	2.5 5.5	99%	0.13 sec 0.51 sec
XOR Arb PUF	4	64 128	Simulation	12 24	99%	3:42 min 2:52 hrs
	5	64 128	Simulation	80 500	99%	2:08 hrs 16:36 hrs
	6	64 128	Simulation	200 —	99%	31:01 hrs —
Lightweight PUF	4	64 128	Simulation	12 500	99%	1:28 hrs 59:42 min
	5	64 128	Simulation	300 1000	99%	13:06 hrs 267 days
	6	64 128	Simulation	— —	99%	— —

TABLE I

State of the Art of modeling attacks on Arbiter PUFs, XOR Arbiter PUFs, and Lightweight PUFs, taken from [32], [34]. All shown results were obtained on simulated CRPs generated by the additive linear delay model [13], [32], [34]. The training times are calculated as if the ML experiment was run on only *one single* core of *one single* processor. Use of k cores will approximately reduce them by $1/k$.

correction method for our side channel that keeps noise levels particularly low. Our approach allows us to attack the two above PUFs for eight single parallel Arbiter PUFs (i.e., for "8 XORs"), and for bitlengths 128, 256 and 512. These sizes had been explicitly suggested as secure in earlier works without implementing them [32], [34]. To our knowledge, our FPGA-implementations of XOR Arbiter PUFs and Lightweight PUFs with 8 XORs and up to 512 bits are the first with comparably large sizes and complexities. We implement these regimes for the first time, and subsequently attack them successfully by our new methods.

Our attacks in practice require physical access to the PUF, which is part of the established Strong PUF attack model [32], [34]. Compared to earlier work [32], [34], they involve the collection of a relatively mild amount of CRPs, and lead to very short computation times. In fact, we empirically estimate the required CRPs to be linear in the size of the PUF (i.e., in the number of XORs and the bitlength), and the computation times to be low degree polynomial. This is in stark contrast to the estimated exponential CRP requirements and computation times of earlier methods on the two above PUF types [32], [34].

Our two side channels are the first power and timing side channel on PUFs discussed in the literature. They are also the first general side channels on Strong PUFs that can increase attack performance (compare above). Both tell an attacker the cumulative number of zeros and the cumulative number of ones in the internal outputs of all the single, parallel Arbiter PUFs within an XOR Arbiter PUF or Lightweight PUF structure. Our power side channel draws its information from closely monitoring the power consumption of the PUF, and from tracing the loads that arise when the final latches that act as "arbiters" in Arbiter PUF implementations switch to one or remain zero. Our timing side channel is based on marking different response patterns with corresponding timing signatures. With the extracted timing signature, we can reversely conclude the composition of multiple individual responses (before the XOR function), deriving the cumulative number of zeros and the cumulative number of ones.

Besides the above new side channels and the first silicon implementation of Arbiter PUF variants with 8 XORs and bitlength up to 512, a fourth central contribution of this paper is the development of a novel, tailormade non-linear regression algorithm. Our algorithm can efficiently exploit the above side channel information on the cumulative number of zeros and ones. To this end, a differentiable model of the side channel information was developed. This new algorithm strongly reduces the complexity of the ML problem associated to XOR Arbiter PUFs and Lightweight PUFs: From the exponential complexity of earlier algorithms *without* side channel information [32], [34] to a linear CRP consumption and low-degree polynomial runtimes in this paper.

At the end of this work, we discuss countermeasures against our side channel attacks, which could and should be put in place in future Arbiter PUF generations. The differential output architecture that we suggest for thwarting power side channels even appears to have applications elsewhere, for example in detecting or correcting output errors in Arbiter PUFs. We also discuss several possibilities to encounter timing side channels, most importantly a method for the construction of an isochronous hardware.

c) Organization of this Paper: Section II provides some background and methology. Sections III and IV illustrate the principles of our power and timing side channels, respectively. Section V shows how existing machine learning methods can be adapted to digest side channel information. Section VI gives the results of applying the adapted methods to simulated CRPs and side channel info. Section VII leads a full proof of concept for combined modeling and timing side channel attacks on FPGA implementations of XOR Arbiter PUFs and Lightweight PUFs. Section VIII describes countermeasures against our side channels. Section IX concludes the paper.

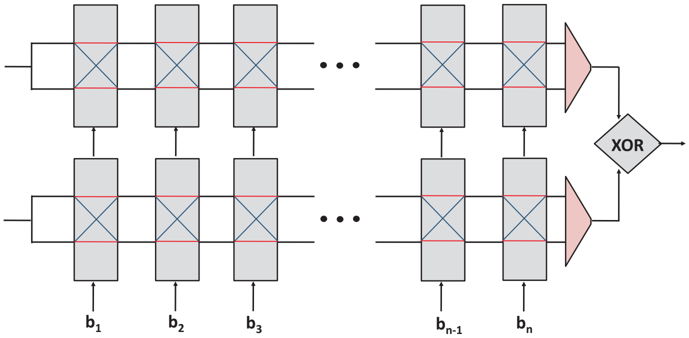

Fig. 1. An XOR Arbiter PUF composed of two single, parallel Arbiter PUFs, also referred to as "2-XOR Arbiter PUF" or "Arbiter PUF with two XORs". The 1-bit outputs of the two single arbiters (red triangles) enter a final XOR gate to produce a 1-bit overall output. The shown structure has bitlength n.

II. Background and Some ML Methodology

A. Arbiter PUF Variants

Arbiter PUFs and variants thereof were among the first electrical PUFs [6], [7], [12], [35], and are currently among the most widespread and best investigated PUF designs. The three variants relevant for this work are described below.

d) Arbiter PUFs: The basic, standard Arbiter PUF has been introduced and discussed in [7], [12], [35]. It consists of a sequence of n stages, for example multiplexers. Two electrical signals race simultaneously and in parallel through these stages. Their exact paths are determined by a sequence of n external bits $b_1 \cdots b_n$ applied to the stages, whereby the i-th bit is applied at the i-th stage. If $b_i = 0$, then the paths run "in parallel" through the multiplexers, and if $b_i = 1$, they cross each other and change position. 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., $C = b_1 \cdots b_n$, and the output of the arbiter element is interpreted as their response R. See [7], [12], [35] for further details. The parameter n is often referred to as the bitlength of the Arbiter PUF.

e) XOR Arbiter PUFs: One possibility to strengthen the resilience of arbiter architectures against machine learning attacks, which has been suggested in [13], [35], is to employ k individual Arbiter PUFs in parallel, each with n stages (i.e., each with bitlength n). The same challenge C is applied to all of them, and their individual outputs r_i are XORed in order to produce a global response o_{XOR}. We denote such an architecture as *"k-XOR Arbiter PUF"* or as *"XOR Arbiter PUF with k XORs"*. The case of a 2-XOR Arbiter PUF is illustrated in Figure 1.

f) Lightweight PUFs: Another type of delay-based PUF, which is often termed Lightweight Secure PUF or Lightweight PUF for short, has been introduced in [16]. It is similar to the XOR Arb-PUF of the last paragraph. At its heart are k individual standard Arb-PUFs arranged in parallel, each with n stages (i.e., with bitlength n), which produce k individual outputs $r_1, \ldots, r_k$. These individual outputs are XORed to produce a multi-bit response $o_1, ..., o_m$ of the Lightweight PUF, according to the formula

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

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 Arbiter PUFs lies in the k inputs $C_1 = b_1^1 \cdots b_n^1$, $C_2 = b_1^2 \cdots b_n^2$, $\ldots$, $C_l = b_1^l \cdots b_n^l$ which are applied to the k individual Arbiter PUFs. Contrary to XOR Arbiter PUFs, it does not hold that $C_1 = C_2 = \ldots = C_k = C$, 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 Arbiter PUF. We refer the reader to [16] for further details.

From the many possible variants of Lightweight PUFs that are enabled by the above parameters m, x and s, we concentrate on the following version in this paper: We consider Lightweight PUFs composed of k parallel standard Arbiter PUFs, and apply the standard input mapping described in [16]. For the output, we take the XOR of all responses of the k parallel Arbiter PUFs. In terms of the bitwise security of the output, this is the most secure variant of the Lightweight PUF: XORing less than k single responses for one response leads to a worsened bit security of the output. For these reasons, exactly the same variant of the Lightweight PUF has been considered in earlier works on the topic [32], [34]. Sticking with this variant furthermore ensures comparability of our data and earlier results [32], [34].

B. Logistic Regression with RProp

From earlier work [32], [34], it is known that an adapted form of logistic regression (LR) performs best on the three PUFs of the last Section II-A. LR is a well-investigated supervised machine learning framework, which has been described, for example, in [1]. In its application to PUFs with single-bit outputs, each challenge $C = b_1 \cdots b_k$ is assigned a probability $p\,(C, r \,|\, \vec{w})$ that it generates a output $r \in \{0, 1\}$). The vector $\vec{w}$ encodes the relevant internal parameters, for example the particular runtime delays, of the individual PUF. The probability is given by the logistic sigmoid $\sigma(x) = (1 + e^{-x})^{-1}$ acting on a function $f(\vec{w}, C)$ parametrized by the vector $\vec{w}$ as

$$p\,(C, r \,|\, \vec{w}) = r\sigma(f) + (1 - r)(1 - \sigma(f)) \tag{2}$$

. Thereby the decision function 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:

$$\hat{\vec{w}} = \mathsf{argmin}_{\vec{w}} \;\; l(\mathcal{M}, \vec{w}) = \mathsf{argmin}_{\vec{w}} \sum_{(C,\,r) \in \,\mathcal{M}} -ln\, p\,(C, r \,|\, \vec{w}) \tag{3}$$

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

$$\nabla l(\mathcal{M}, \vec{w}) = \sum_{(C,\, r) \in \mathcal{M}} \left(\sigma\left(f\left(\vec{w}\right)\right) - r\right) \nabla f(\vec{w}) \tag{4}$$

From the different possible optimization methods, RProp [1] [25] has been identified as optimal in earlier ML works on PUFs [32], [34]. RProp makes a very big difference in convergence speed and stability of the LR algorithms (k-XOR Arbiter PUFs for medium or large k were only learnable with RProp).

In general, 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 support vector machines, for example, but merely differentiable.

C. Linear Additive Delay Model

It has become standard to describe the functionality of Arbiter PUF variants via an additive linear delay model [13], [32], [34]. The overall delays of the two racing signals are modeled as the sum of the delays in the stages. The final delay difference Δ between the upper and the lower path in an n-bit Arbiter PUF is expressed as

$$\Delta = \vec{w}^T \vec{\Phi}, \tag{5}$$

where $\vec{w}$ and $\vec{\Phi}$ are vectors of dimension $n+1$. The parameter vector $\vec{w}$ encodes the delays for the subcomponents in the Arbiter PUF stages, whereas the feature vector $\vec{\Phi}$ is solely a function of the applied n-bit challenge C [13], [32], [34].

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

$$\vec{w} = (w^1, w^2, \dots, w^k, w^{n+1})^T, \tag{6}$$

where $w^1 = \dfrac{\delta_1^0 - \delta_1^1}{2}$, $w^i = \dfrac{\delta_{i-1}^0 + \delta_{i-1}^1 + \delta_i^0 - \delta_i^1}{2}$ for all $i = 2, \dots, n$, and $w^{n+1} = \dfrac{\delta_n^0 + \delta_n^1}{2}$. Furthermore,

$$\vec{\Phi}(\vec{C}) = (\Phi^1(\vec{C}), \dots, \Phi^k(\vec{C}), 1)^T, \tag{7}$$

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

The output r of an Arb-PUF is determined by the sign of the final delay difference Δ:

$$r = \Theta(\Delta) = \Theta(\vec{w}^T \vec{\Phi}). \tag{8}$$

with Θ being the Heaviside step function, i.e., $\Theta(x) = 0$ if $x < 0$ and $\Theta(x) = 1$ if $x \geq 0$. Eqn. 8 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 $r = 0$, those with feature vectors on the other side $r = 1$. Determination of this hyperplane allows prediction of the PUF and can be achieved by setting the decision function f in Eqn. 2 to the linear delay model $f = \vec{w}^T \vec{\Phi}$.

More complex architectures that use k standard Arbiter PUFs in parallel, possibly together with special input or output mappings, can then simply be modelled by using the linear additive delay model for each of the parallel Arbiter PUFs. Overall, this involves k feature vectors $\vec{\Phi}_1, \dots, \vec{\Phi}_k$ derived from the effective challenges at the individual Arbiter PUFs (given by the input mapping) and k weight vectors $\vec{w}_1, \dots, \vec{w}_k$:

$$o = \Theta(\prod_{i=1}^{k} \Delta_i) = \Theta(\prod_{i=1}^{k} \vec{w}_i^T \vec{\Phi}_i) \tag{9}$$

Eqn. 9 defines a decision boundary at $\prod_{i=1}^{k} \vec{w}_i^T \vec{\Phi}_i = 0$. Any challenges C that have their feature vector set $\vec{\Phi}_1, \dots, \vec{\Phi}_k$ located on the one side of the boundary (e.g. $o < 0$ respectively an odd number of individual delays Δ_i smaller than zero) give response $r = 0$, those with feature vectors on the other side $r = 1$. Determination of this boundary allows prediction of the PUF and can be achieved by setting the decision function f in Eqn. 2 to this boundary $f = \prod_{i=1}^{k} \vec{w}_i^T \vec{\Phi}_i$. It implies an optimization along the gradient in Eqn. 4:

$$\nabla f(\vec{w}_j) = \vec{\Phi}_j \prod_{i \neq j} \vec{w}_i^T \vec{\Phi}_i \tag{10}$$

This principle has been applied to the XOR Arbiter PUF and Lightweight PUF of this paper.

D. Numerical CRP Generation, Simulated Side Channels

A subset of the attacks presented in this paper are executed on simulated CRPs generated by the additive linear delay model of the last section. The simulated challenge-response pairs (CRPs) were generated in the following fashion: (i) The delay values for all single Arbiter PUFs within the respective PUF variant 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 Eqn. 5, it amounts to choosing the $n + 1$ entries of the vector $\vec{w}$ pseudo-randomly according to a standard normal distribution. (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). The delays of the two electrical signal paths are simply added up and compared. Or, again in the language of of Eqn. 5, the value of Δ is computed.

In a subset of our attacks, also simulated side channel information is used. The cumulative number of ones in the outputs of the single Arbiter PUFs was obtained directly from the above CRP simulation in the linear additive delay model, i.e., simply by adding up the outputs of the single, parallel Arbiter PUFs in the above CRP simulation. An analog statement holds for the cumulative number of zeros.

E. Training Set, Test Set, and Prediction Error

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 ML experiments throughout this paper, each test set consisted of 10,000 randomly chosen CRPs. The size of the employed training sets varies, and is given individually on each occasion. The prediction rate is $1 - \epsilon$.

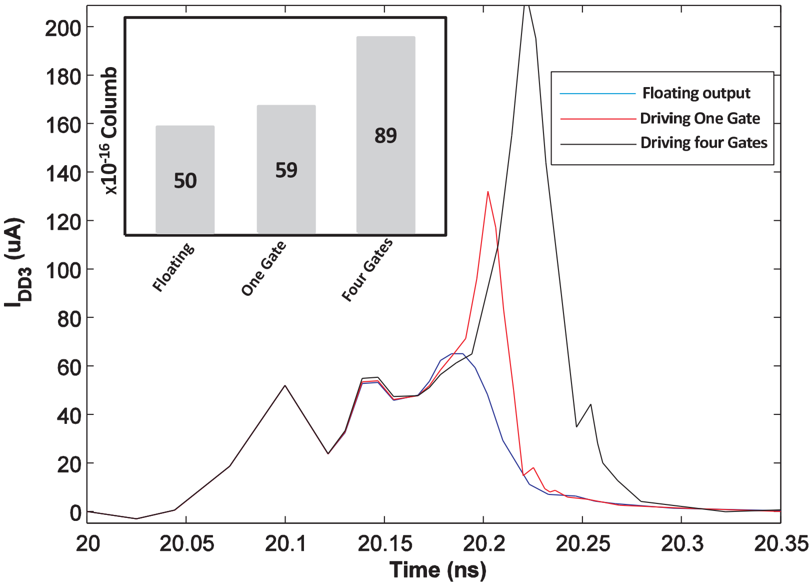

Fig. 2. The power tracking side channel analysis for a latch that had a transition to 1, with different driving loads, in SPICE simulation. The inset is the amount of drawn charges, which is calculated from the area under each curve. The amount of charges is linearly proportional with the number of gates, noting that the amount of charges normally drawn for a floating load should be subtracted.

The term N_{CRP} (or simply *"CRPs"*) denotes the number of CRPs employed in an attack. In the case of logistic regression (LR), which is the only ML algorithm employed in this paper, this number is always equal to the size of the training set.

F. Employed Computational Resources and Training Times

We used an Intel Xeon X5650 processor at 2.67GHz with 48 GB of RAM in all of our ML experiments, representing a worth of a few thousand Euros. All computation times (= "training times") are calculated for one core of one processor of this hardware.

III. Power Side Channels on Arbiter PUF Variants

The basic concept of our power side channel is to apply power tracing to determine the transition from zero to one of the latches (=arbiter elements) that are part of Arbiter PUF based architectures. The power tracing technique is based on measuring the amount of current drawn from the supply voltage during any latch transition to one.

In order to validate our approach, we implemented a SPICE simulation that uses only one latch with three different outputs loading (floating output, output connected to one gate, and output connected to four gates). Figure 2 illustrates our results, and shows the different amount of current drawn for the three different output loading. The reason for having different values for the different loading is that an additional amount of charges is required to charge the capacitance of each gate. Hence, the amount of drawn charges, which is the integration of the current curve, is linearly proportional with the number of gates. Taking into consideration, the amount of charges normally drawn in case of a floating load should be subtracted. Consequently, for extreme cases when all latches' output in the device are zeros or all are ones, this power tracing technique would allow the attacker to determine these cases very easily.

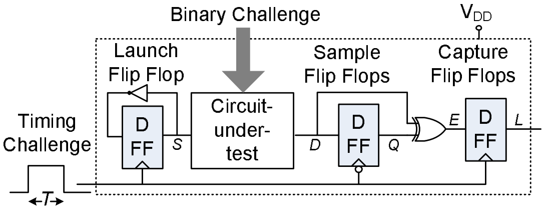

Fig. 3. The timing signature extraction circuit.

By applying this idea to XOR Arbiter PUFs and Lightweight PUFs, which utilize more than one single Arbiter PUF in parallel, one could determine the cumulative number of ones (and zeros) stored in the latches (arbiters) within the PUF. Following our above discussion, the power consumption will tell us the cumulative number of zeros and ones that are stored in the latches (or, in other words, the cumulative number of zeros and ones that is output by the single Arbiter PUFs before the large, final XOR gate. Please note, however, that we cannot derive which of the single Arbiter PUFs in an XOR Arbiter PUF have output one and which have output zero. We only know the overall number of zeros and ones before the XOR. Taken by itself, the side channel thus appears relatively worthless. We will nevertheless show that this basic information can boost PUF attacks significantly if it is used in the right manner in Section V.

IV. Timing Side Channels on Arbiter PUF Variants

A. Delay Measurement Circuitry

Let us now describe our timing side channel approach, starting with our delay measurement circuit. It operates on the basis of FPGA reconfigurability. Figure 3 demonstrates an abstraction of the delay measurement concept and circuitry. To measure the timing of the circuit under test (CUT), three on-chip flip flops (FFs) connected to the chip clock are utilized: *launch*, *sample*, and *capture FFs*. For now, assume that the CUT input is held the same and therefore, the circuit delay is fixed.

The clock signal with a period of *2T* is applied to the CUT when all the FFs are set to an initial value. For the sake of explanation, assume that the initial value of each FF is *zero* and the CUT's output, after applying the input, is expected to change from *zero* to *one*. At the clock's rising edge, the launch FF sends a low-to-high signal to the CUT. At time T, i.e., half the clock period, the sample FF samples the output at the falling clock edge. (Note the inverted clock signal at the sample FF.)

There are three possible scenarios for the value of the sample FF: (i) the CUT delay is less than T, in which case the sample FF will be set to the correct value (*one*); (iii) the CUT delay is (almost) equal to T and therefore the FF shall be in a transition mode; and (ii) the CUT delay is greater than T and thus the sample FF will not be set to the final value. As shown on the Figure, an XOR gate compares the values of the actual output and the sample FF. The capture FF would hold the comparison results for one clock cycle. In case (i), the comparison would demonstrate equal values of the two XOR inputs, while in case (iii) the two XOR inputs would differ, generating an error signal (one) at the capture FF. Case (ii) is trickier to predict, as the output signal in transition may be interpreted as zero or one at the input of the XOR gate with a certain probability.

We now perform a more careful timing analysis. Let t_{cut}, t_{c2Q}, t_{skew}, t_{setS}, t_{holdS}, t_{setC}, t_{holdC}, and t_{xor} denote the values for the CUT, clock-to-Q of the launch FF, clock skew between the launch and sample FFs, the sample FF's setup, the sample FF's hold, the capture FF's setup, the capture FF's hold, and XOR timings respectively. If t_p denotes the total propagation delay from the moment the launch FF is clocked to the moment the signal passes through CUTs and settles at the sample FF, then $t_p = t_{cut} + t_{c2Q} - t_{skew}$.

In case of near equality of t_p and T, i.e., scenario (ii) above, because of the setup and hold timing violations the sample FF would enter a metastable point. The probability of the metastable point resolving to a zero or one value at the capture FF is a function of the proximity of t_p and T. As an example, in case of having equal T and t_{cut} both clock and signal simultaneously arrive at the sample FF. Thus, the metastable point resolves to a *zero* or *one* with equal probability.

When the XOR output at the capture FF does not demonstrate a discrepancy, the following relationship holds:

$$t_{holdC} < t_p < T - t_{setS} \tag{11}$$

As t_p enters the following interval,

$$T - t_{setS} < t_p < T + t_{holdS} \tag{12}$$

the capture FF shows the output metastability.

Note that the probability of occurrence of timing errors demonstrates a periodic rise and fall pattern. First, the probability of timing error rises as t_p gets closer to the upper bound of Inequality above. The timing error would occur every cycle (with a high probability) once the following condition holds:

$$T + t_{holdS} < t_p < 2T - (t_{setC} + t_{xor}) \tag{13}$$

Next, once the value of t_p exceeds the $2T - (t_{setC} + t_{XOR})$ in Equation 13, the timing error's rate starts to decline. The fall in error probability is not because of the correctness of functionality, but it is because of the limitations of the capture FF in observing and recording the timing errors.

Equation 14 (below) demonstrates the transition from the scenario where there is a high probability of error in each clock cycle (Equation 13) to the scenario where no errors shall be detected (Equation 15).

$$2T - (t_{setC} + t_{xor}) < t_p < 2T + (t_{holdC} - t_{xor}) \tag{14}$$

$$2T + (t_{holdC} - t_{xor}) < t_p < 3T - t_{setS} \tag{15}$$

In case t_p is larger than $3T - t_{setS}$, the timing errors shall not be hidden anymore. Indeed, the timing errors emerge and could be captured if t_p is within the interval below:

$$3T - t_{setS} < t_p < 3T + t_{holdS} \tag{16}$$

Assuming that Inequality 17 holds, the timing errors would be detectable in each clock cycle:

$$3T + t_{holdS} < t_p < 4T - (t_{setC} + t_{xor}) \tag{17}$$

This rise and fall pattern repeats for integer multiples of T; the maximum FPGA clock frequency sets an upper bound on this behavior. If the value of $T >> t_{xor}$ and $T >> t_{set}$ and $T >> t_{hold}$ (for the FFs), it is possible to approximate the intervals as $n \times T < t_p < (n+1) \times T$ where timing errors shall be detected for odd n values.

Note that in Figure 3, the propagation delay of the CUT circuit may be different, in practice, for the low-to-high and high-to-low transitions. Suppose that one of them is greater than the other, e.g., $t_p^{l \to h} < t_p^{l \to h}$. In this case, the $t_p^{h \to l}$ satisfies the Equation 13. The timing errors for this scenario only occur for the high-to-low transition and therefore, the errors can be observed only half of the times. The final measurement shows a superposition of both transition effects.

Figure 4 demonstrates the measured probability of timing error with respect to T in the top plot. Inequality 13 drives the pattern seen in region R_5. The metastability in inequality 14 is reflected in regions R_6 and R_8. The error-free area R_9 corresponds to Inequality 15. Regions R_3, R_7 and R_{11} are results of the difference between $t_p^{l \to h}$ and $t_p^{l \to h}$. Inequality 16 is reflected in regions R_{10} and R_{12}. Lastly, region R_{13} depicts the Equation 17.

Note that all the delays defined in this section, i.e., for the FFs, clock skews, and XOR, shall have two different values for the rising and falling edge transitions. The inequalities defined in this section hold for both rising and falling transitions.

In the remainder of this paper, we refer to the timing characterization circuit in Figure 3 as a *timing characterization cell* or simply a *cell*. In our implementation, we place each cell in one configurable logic block (CLB) on FPGA.

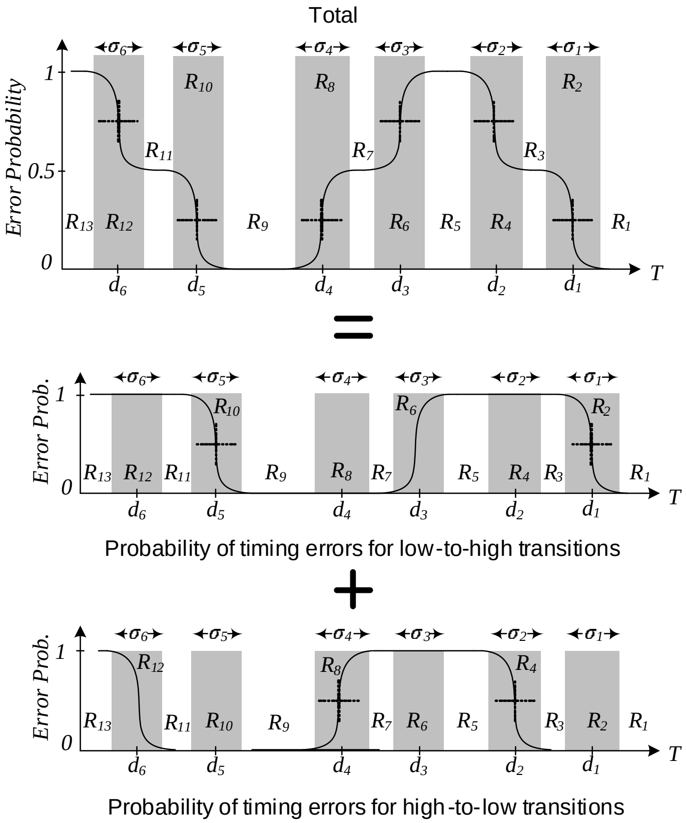

Fig. 4. Probability of incurring timing errors as a function of half clock period (T).

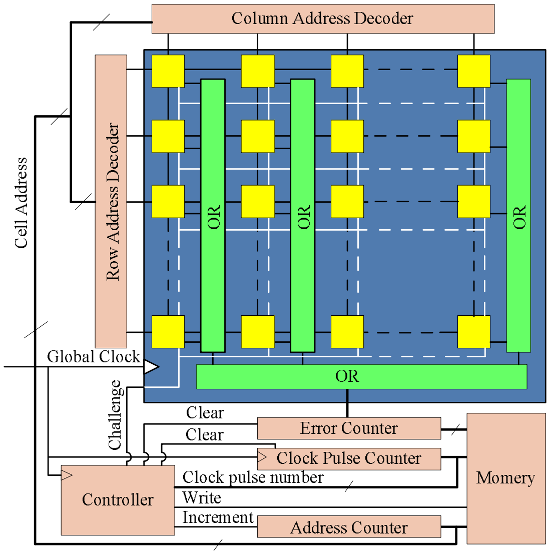

Fig. 5. The architecture for chip level delay extraction of logic components.

B. Timing Characterization Method

We now describe the timing measurement system which efficiently extracts the probability of obtaining timing failure for various clock pulse widths on the FPGA. The circuit demonstrated in Figure 3 only generates a single bit flag indicating the presence or absence of an error. Our objective is to design a method to measure the rate or probability at which errors appear at the circuit output in Figure 3 to extract the transitions as discussed in Figure 4.

In order to measure the probability of error at a certain clock frequency, an error histogram accumulator is realized by two counters. The first one is an error counter whose value increments by one each time an error occurs. The second one counts the clock cycles and also after 2^N clock cycles clears (resets) the error counter and restarts again, where N is the binary counters' size. The error counter value is stored in the memory one clock cycle before it is reset. Now, the stored number of flaws normalized to N would yield the error probability value.

Next, we linearly and continually sweep the input clock frequency: in T_{sweep} seconds from $f_i = \frac{1}{2T_i}$ to $f_t = \frac{1}{2T_t}$, where $T_t < t_p < T_i$. For each frequency sweep a separate set of registers count the number of clock pulses. We use this counter as an accurate timer which records the frequency of the timing errors. This counter value is retrieved every time the content of the error counter is written into memory. The value of this counter is retrieved every time the error counter content is written into memory.

The system described above is utilized for extracting the delays of any CUT implemented on FPGA. A logic configuration can be used within the CUT in the characterization circuit. The delay of each CUT component can be found by sweeping the clock frequency once. Note that the scanning for extracting delay values could also be performed in parallel to reduce the characterization time .

C. Characterization Accuracy

The resolution of the delay measurement, i.e., the measured delay's accuracy, is a function of a few factors: (i) the clock noise and skew, (ii) the sweeping frequency resolution, and (iii) the number of pulses at each frequency. The output of the characterization circuit is a binary zero/one value. A real-valued output can be measured by repeating several (same width) clock pulses to the circuitry and accumulating the number of ones at the output. The resulting value, when normalized, shows the probability at which the timing errors occur for each input clock's pulse width. The more the input clock pulse is repeated, the higher sampling resolution and accuracy can be achieved.

Next, assume that the clock pulse (of width T) is sent to the CUT for M times. Because of clock skew and phase noise, the characterization circuitry receives a clock pulse with width $T_{eff} = T + T_j$, where T_j is the additive jitter. Suppose that T_j is a random variable with a zero mean and symmetric distribution around its mean. The output probability is a continuous and smooth function of T_{eff}; thus, approximating the probability by averaging shall be an asymptotically unbiased estimator as $M \to \infty$. Lastly, the minimum measurable timing is a function of the maximum clock speed at which the FFs can be run (maximum clock frequency). During a linear frequency sweep, a longer sweep time increases both items (ii) and (iii) and thus the characterization accuracy.

D. Parameter Extraction

Thus far, we have described a system that measures the probability of timing errors for various clock pulse widths. The error probability can be fully represented by a set of few parameters; the parameters are directly related to the CUT delay and FF setup and hold times. It can be shown that the probability of timing errors shall be written as the sum of shifted Gaussian CDFs. The central limit theorem can determine the Gaussian nature of the error probabilities which can be explained by Equation 18 shows the parameterized error probability function.

$$f_{\mathbf{D},\Sigma}(t) = 1 + 0.5 \sum_{i=1}^{|\Sigma|-1} -1^{\lceil i/2 \rceil} \left[Q(\frac{t-d_i}{\sigma_i}) \right] \tag{18}$$

where $Q(x)$=$\frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\frac{u^2}{2}\right)$ and $d_{i+1} > d_i$. To estimate the timing parameters, f is fit to the set of measured data points (t_i,e_i), where e_i is the error value recorded when the pulse width equals t_i.

E. Side-channel timing analysis of XOR'ed outputs

The objective of the timing circuitry and measurement method (described above) is for providing additional information about the individual response bits (i.e., PUF output bits) even though the response bits are XOR'ed together for providing the output. Assume that k response bits $\{r_1, \ldots, r_k\}$ are XOR'ed to form a single output bit o. (Note that a k-input XOR shall consist of several stages of smaller XOR gates. For the same of demonstration, assume that the delay of the response bit r_i, denoted by t_{r_i} follows a certain order, say $t_{r_1} \leq t_{r_2} \cdots \leq t_{r_{k-1}} \leq t_{r_k}$).

As we sweep the clock frequency (in a rising trend) to measure the XOR'ed response bit, we eventually get to a regime where the frequency of the sweeping clock is close to the overall output (after XOR'ing). However, before we get to that regime, there are clock periods for which only a few XOR inputs (i.e., response bits) change. Sweeping the clock frequency could yield the information about the approximate timing of the XOR inputs. Even though we would not be able to tell which of the response bits had changes, we shall, with a good probability, determine the number of flipping XOR inputs. This number shall be vague if the timings of two or more response bits coincide. Since the probability of such a coincidence is rather low, in most instances clock sweeping can give us an approximation of the number of flipped XOR inputs, i.e., on the cumulative number of zeros and ones among the single Arbiter PUF responses $r_1, \ldots, r_k$.

V. Combining Our Side Channel Information with Machine Learning Techniques

The question if (and how) the side channel information on the cumulative number of zeros and ones can be efficiently exploited in modeling attacks turned out to be non-trivial. The obvious problem is that this number does not indicate *which* of the single Arbiter PUF outputs has been zero or one. In a first attempt to resolve the problem, we only used those CRPs where the side channel indicates that all single Arbiter PUF outputs are one or that all are zero in the modeling process. In these cases, we obviously *do* know every single output. By collecting such *"special"* CRPs, the problem of modeling an entire Lightweight PUF decomposes into the problem of modeling single Arbiter PUFs, which is known to be simple, and can be done with a linear number of CRPs and an approximately quadratic computational complexity [32], [34]. However, one of several problems with this strategy is that the *"special"* CRPs are exponentially rare, and merely constitute a fraction of $1/2^{k-1}$ of all CRPs. One therefore needs to collect an exponential amount of CRPs to implement this strategy.

Finally, we found a gradient based optimization similar to the logistic regression (LR) algorithm that has been used in earlier ML attacks on the XOR Arbiter PUF and Lightweight PUF [32], [34], and which is described in Section II-B. The following treatment assumes some familiarity with this algorithm and with the work in [32], [34].

Let $r_i(C) \in \{0,1\}$ be the output of the i^{th} Arbiter PUF within a k-XOR Arbiter PUF (or within a Lightweight PUF with k parallel Arbiter PUFs) to a challenge C. The side channel information then yields the number n of individual Arbiter PUFs with output one: $n = \sum_i r_i(C)$. It lies in contrast to the general setting of binary outputs in LR on an interval scale. Therefore, instead of optimizing the binary class probabilities Eqn. 2, we rely on minimizing the squared error between a side channel model $f(\vec{w}, C)$ and the actual outputs n:

$$l(\mathcal{M}, \vec{w}) = \sum_{(C,\,t) \in \mathcal{M}} (f(\vec{w}, C) - n)^2.$$

The corresponding gradient

$$\nabla l(\mathcal{M}, \vec{w}) = \sum_{(C,\,r) \in \mathcal{M}} 2\left(f\left(\vec{w}\right) - n\right) \nabla f(\vec{w}) \tag{19}$$

is highly similar to the gradient in LR (Eqn. 4) and we will again apply the RProp update scheme (as in [32], [34]) to find a solution $\hat{\vec{w}}$ with minimal error l.

Assuming the standard linear additive delay model (see Section II-C and Eqn. 5), one obtains the following model of the side channel information:

$$f(\vec{w}, C) = \sum_i \Theta(\vec{w}_i^T \vec{\Phi}_i).$$

Note that the model only depends on the direction, but not on the length $\|\vec{w}_i\|$ of the weight vectors. That is, any two solutions $\vec{w}_i$ and $\alpha\vec{w}_i, \alpha \in \mathbb{R}^+$ are equivalent. Therefore we might substitute the Heaviside function by the differentiable logistic sigmoid $\sigma(x) = (1 + e^{-x})^{-1}$ to enable gradient based optimization. This is a reasonable substitution as $\lim_{\|\vec{w}\| \to \infty} \sigma(\vec{w}^T \vec{\Phi}) = \Theta(\vec{w}^T \vec{\Phi})$ and, as noted above, a valid solution is unaffected by scaling of $\vec{w}$.

As this substitution makes the model differentiable, we obtain the following gradient to insert in Eqn. 19:

$$\nabla f(\vec{w}_j) = \sigma(\vec{w}_j^T \vec{\Phi}_j)(1 - \sigma(\vec{w}_j^T \vec{\Phi}_j))\vec{\Phi}_j. \tag{20}$$

This gradient of an individual Arbiter PUF's weight vector $\vec{w}_j$ depends only on the value of the weight vector itself. It is in contrast to the gradient of Eqn. 10, which does depend on the weight vectors $\vec{w}_i$ of all other Arbiter PUFs. The decoupling of individual Arbiter PUF updates thus drastically simplifies the ML problem, provided that side channel information is available.

In addition to the above new regression, we applied a two step optimization methodology: First we optimized the PUF model based on the above process and gradient, using the side channel information, until a fraction of $f = 0.95$ percent of the final XOR Arbiter output was correctly reproduced. Secondly, we further refined and optimized the model with the "standard" LR algorithm applied in [32], [34] for 1000 iterations. This led to very low error rates around 2% or below. For all experiments, we used hundred times more CRPs than free parameters in the model, i.e.,

$$N_{CRP} \approx 100 \times \text{bitlength} \times \text{no. of XORs}.$$

Note that the above equation merely describes a linear CRP consumption in the problem parameters. This is in stark contrast to the exponentially growing complexities of pure machine learning attacks on XOR Arbiter and Lightweight PUFs (see Table I and [32], [34]).

While our above process in the first step of the above methodology mostly converged to the global minimum, in a few cases it got stuck (i.e., the performance after 5000 iterations was worse than 5% remaining missclassifications). In this case, we restarted the algorithm with a different random initialization of $\vec{w}$.

VI. Results on Simulated CRPs

We now analyze the performance of our combined modeling and side channel technique on simulated, noise-free CRPs and side channels, as detailed in Section II. As discussed earlier, simulated CRPs generated by the linear additive delay model are very close to silicon CRPs of Arbiter PUF variants, as proven in [34]. A very detailed discussion on the use of simulated CRPs, PUF noise, and ML performance, which advocates and justifies the use of simulated CRPs in security analyses of any Arbiter PUF variants, is given Section II-G of Rührmair et al. [34].

All results of our ML algorithms on simulated CRPs and side channel information are given as data points on a logarithmic scale in Figure 6. They reach from 2 to 16 XORs with a stepwidth of 1, and concern 64, 128, 256 and 512 bits. For selected values, the detailed Figures are given in Table II. Note that the side channel information on the cumulative number of zeros and ones is the same for both our power and timing side channel. The results of the table therefore apply to both side channel approaches.

Table II and Figure 6 shows that the training times and CRP consumptions of our approach are remarkably small, especially compared to the exponentially growing complexities of pure machine learning attacks on XOR Arbiter and Lightweight PUFs (see Table I and [32], [34]). The number of CRPs that we used in all cases followed the earlier formula

$$N_{CRP} \approx 100 \times \text{bitlength} \times \text{no. of XORs}.$$

The only exception were the cases of 14 XORs and 16 XORs for bitlengths 512, where our RAM was insufficient to carry all CRPs. In these cases, we used $50 \times \text{bitlength} \times \text{no. of XORs}$ (see Table II). This led to slightly longer computation times, but did not prevent successful learning. Once more, this indicates the robustness of our method against small parameter changes.

We also thoroughly analyzed the training times of our algorithm. The results are illustrated on a logarithmic scale in Figure 6. The data shows that the computation times (=training times) are low-degree polynomial, presumably cubic, in the problem size (i.e., in the bitlengths and the number of XORs). Therefore our attacks can, in principle, be extended quite easily to larger PUF sizes. At the same time, the sizes of XOR-based Arbiter PUFs cannot be rised indefinitely: The practical instability of the XOR Arbiter PUF and the Lightweight PUF increases exponentially with the number of XORs, as already observed in [32], [34]. The two considered PUF designs hence can no longer be regarded secure without countermeasures in the presence of side channel attacks.

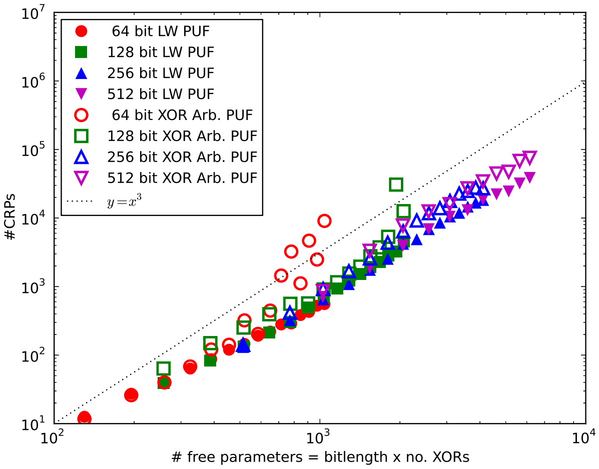

Fig. 6. The training times for our ML-algorithm on Lightweight PUFs (LW PUFs) and XOR Arbiter PUFs on a logarithmic scale. They show that the computational complexity regarding training times is cubic, i.e., $O(x^3)$.

A final interesting effect is that with side channel information, the performance of our ML algorithms on Lightweight PUFs is slightly faster than for XOR Arbiter PUF, leading to smaller computation times. Without side channels, the converse effect has been observed [32], [34]. Intuitively, the challenge input mapping of the Lightweight PUF creates a more diverse and stable information basis for the ML algorithm, which leads to faster convergence. A full mathematical analysis of this effect will be conducted in future work.

VII. Proof of Concept for FPGA Implementations

A. Implementation of Lightweight PUFs and XOR Arbiter PUFs on FPGAs

To verify the feasibility of the proposed methods, both XOR Arbiter PUFs and Lightweight PUFs are built on Spartan-6 FPGAs, using the architecture for Lightweight PUFs described in Section II-A. The basic element for the described PUFs is a single Arbiter PUF, like any of the two single Arbiter PUFs

No. of XORs	Bit Length	Source of CRPs and Side Channel Info	CRPs ($\times 10^3$)	Prediction Rate XOR Arb. PUF	Training Time XOR Arb. PUF	Prediction Rate Lightw. PUF	Training Time Lightw. PUF
8	64	Simulation	51.2	98.6%	5:21 min	98.6%	2:25 min
	128		102	98.7%	15:06 min	98.5%	11:48 min
	256		205	98.5%	1:46 hrs	98.7%	1:10 hrs
	512		410	98.5%	9:31 hrs	98.9%	5:11 hrs
10	64	Simulation	64.0	98.5%	7:25 min	98.2%	3:40 min
	128		128	98.4%	25:49 min	98.5%	20:15 min
	256		256	98.2%	3:16 hrs	98.8%	1:55 hrs
	512		512	98.2%	13:10 hrs	98.8%	6:47 hrs
12	64	Simulation	76.8	98.8%	53:59 min	98.2%	4:49 min
	128		154	98.1%	46:09 min	98.2%	31:54 min
	256		307	98.0%	4:56 hrs	98.5%	2:55 hrs
	512		614	98.0%	21:01 hrs	98.1%	10:42 hrs
14	64	Simulation	89.6	98.4%	1:18 hrs	98.1%	7:06 min
	128		179	98.1%	1:29 hrs	98.2%	47:07 min
	256		358	97.6%	6:54 hrs	98.2%	4:04 hrs
	512		358	97.0%	38:55 hrs	96.8%	7:18 hrs
16	64	Simulation	102	98.2%	2:31 hrs	97.9%	9:14 min
	128		205	97.9%	3:30 hrs	97.9%	1:15 hrs
	256		410	97.7%	7:33 hrs	98.2%	5:06 hrs
	512		410	96.4%	46:28 hrs	96.7%	9:20 hrs

TABLE II
Performance of combined modeling and side channel attacks on XOR Arbiter PUFs and Lightweight PUFs. All shown results were obtained on simulated CRPs generated by the additive linear delay model [13], [32], [34]. The side channel information on the number of zeros and ones before the XOR gate is simulated and noise-free, too; please note that it is the same for power and ti ming side channels (namely the cumulative number of zeros and ones before the XOR gate).

in Figure 1. 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 in Figure7, as suggested by Majzoobi et al. [18], [15].

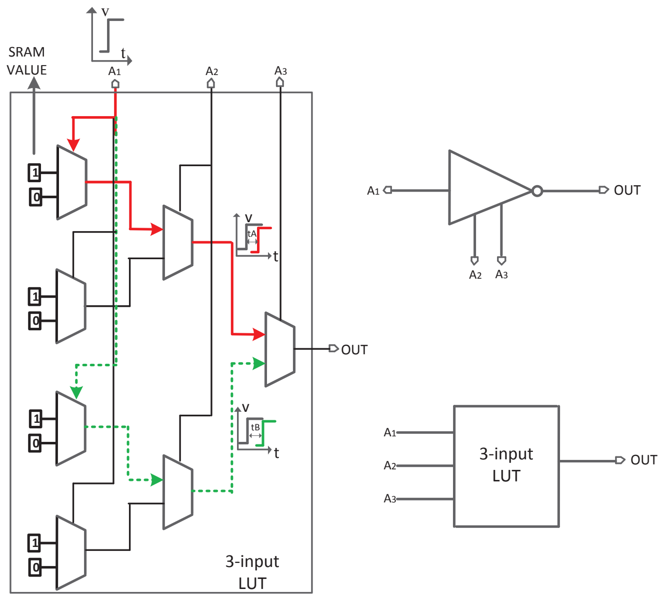

Fig. 7. LUT based Programmable Delay Line

For each CRP, majority voting over five repeated 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 aN-bit pseudorandom 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 Eqn. 21:

$$F = 1 + X^1 + X^3 + X^4 + X^{64} \tag{21}$$

B. Implementation of the Timing Side Channel Measurements

We executed the timing measurements of Section IV on our PUF implementations on Digilent Spartan 6 FPGAs of Section VII-A. Both XOR Arbiter PUFs and Lightweight PUF with 8 XORs and bitlengths 64, 128, 256 and 512 were built. For delay signature extraction, designed PUF circuits are utilized as the CUT part in Figure 3. For each CRP, we collected both side channel info on the cumulative number of ones and zeros, as well as the “global” response o of the PUF after the final XOR gate. The latter response was stabilized by repeating the CRP measurement five times, and by applying majority voting (see above). As indicated above, in most instances clock sweeping only gives us an approximation of the number of flipped XOR inputs (i.e., of the cumulative ones and zeros).

In order to avoid errors in the side channel info, we applied the following procedure: (i) We executed a constency check: If the measured global response o is zero, is the measured number of ones in the side channel info even? Likewise, if o is one, is the number of ones in the side channel odd? If any inconsistencies occured, this CRP was discarded and not used in the ML process. (ii) We decrease the range of each response pattern to make the classification more disambiguous, from 12 ps to 8ps. If the measured side channel data data lay outside these narrowed ranges, the CRP again was not used in ML. After these two steps, 52% of all CRPs could still be used. Table III gives the number of usable CRPs in the CRP column.

No. of XORs	Bit Length	Source of CRPs and Side Channel Info	CRPs ($\times 10^3$)	Prediction Rate XOR Arb. PUF	Training Time XOR Arb. PUF	Prediction Rate Lightw. PUF	Training Time Lightw. PUF
8	64	FPGA	26	98.5%	2 min	98.5%	1 min
	128		51.6	97.5%	12 min	98.2%	9 min
	256		103	97.7%	1:35 hrs	97.8%	1:00 hrs
	512		205	97.4%	16:50 hrs	97.5%	3:30 hrs
12	64	FPGA	39	98.1%	16.5 min	98.5%	2 min
	128		77.4	97.4%	38.5 min	97.9%	24.1 min
	256		154.5	97.1%	3.8 hrs	97.3%	1.75 hrs
	512		308	96.92%	56.25 hrs	97.11%	9.55 hrs
16	64	FPGA	52	98%	37 min	98%	7 min
	128		103.2	97.5%	2 hrs	97.5%	51.7 min
	256		206	97.3%	15.1 hrs	96.9%	4.8 hrs
	512		410	96.5%	102 hrs	96.7%	20.2 hrs

TABLE III
FULL EXPERIMENTAL PROOF OF CONCEPT OF COMBINED MODELING AND TIMING SIDE CHANNEL ATTACKS, CARRIED OUT ON FPGA IMPLEMENTATIONS OF THE XOR ARBITER PUF AND LIGHTWEIGHT PUF.

C. Results

We applied our adapted ML algorithms of Section V to silicon CRP data collected from the above FPGA implementation of Lightweight PUFs and XOR Arbiter PUFs, using silicon side channel information collected directly from these FPGAs (see Sections VII-B and VII).

Our approach led to the results shown in Table III. There is very little performance loss compared to the results for 8 XORs and bitlengths 64, 128, 256, and 512 on simulated data (see Table II). The number of used CRPs followed the formula 50 × bitlength × no. of CRPs, i.e., we only used half as many CRPs as in the silicon case. The good results shows the robustness of our method, and indicate the efficiency of our above measures for the error-free collection of CRPs and associated side channel info. They also settle once more the viability of ML results on simulated CRPs, which was already postulated and confirmed in earlier work [32], [34], even for very large number of XORs and bitlengths, and even in the presence of side channel information. Our result breaks “real”, silicon FPGA implementations of 8 XOR Arbiter PUFs with bitlengths up to 512. The latter construction had been explicitly suggested as secure against pure modeling approaches in earlier work [32], [34].

VIII. COUNTERMEASURES

A. Countermeasures against our Power Side Channel

In order to immunize arbiter-based PUFs against power tracking SCA, one could add an additional, symmetric arbiter at the last stage of the PUF. The input signal for the added arbiter is inverted before it enters the arbiter (see Figure 8).

This idea behind the architecture is obviously to keep the number of zeros and ones in the entire PUF architecture constant, thereby preventing power tracking side channel attacks. A differential output of the described kind could also be desirable in fault tolerant applications, since it might allow error correction, and indicate unstable responses. Recall that the arbiter (=latch element) typically is one of the main sources of instability in an arbiter PUF: Signal pathes with a runtime difference that is on the order of the switching time of the latch often cause unstable responses. If the two differential latches have inconsistent content, this indicates the occurence of an unstable PUF response.

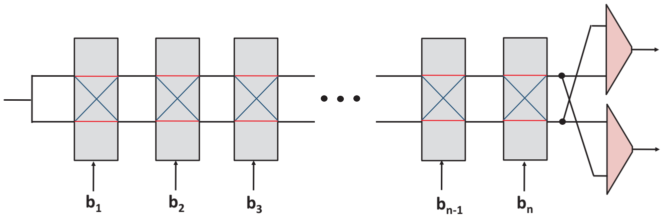

Fig. 8. Standard Arbiter PUF with a differential output. Two arbiters are employed to generate two differential response bits, which have inverted input signals. The architecture would prevent the use of power tracking SCA. Asides, it also provides new error detection and correction capabilities.

Use of k latches in such a differential design, together with standard majority voting over the latches’ outputs, could be used as a simple and very efficient means for response stabilization and error correction. We did not work out this idea in detail, however, since it is not within the main scope of this paper.

B. Countermeasures againt our Timing Side Channel

The timing side channel attacks depend on the correlations between the secret and the information leaked through the side channels. Several different countermeasures appear possible. For example, it is possible to jam the leaked timing side-channel with random pulses after the XOR so the attacher is not able to find the exact switching time of the XOR gates. Another plausible countermeasure is to place XOR/XNOR devices such that every time we get exactly one of them switching so it is hard to know which of the signal directions was the legitimate one.

For the sake of brevity, we concentrated on an effective countermeasure where we devise an isochronous hardware and thus, it runs exactly a constant amount of time and is independent of secret values. The reason we focus on this one is that the hardware overhead is rather small. In this countermeasure, we place a 2-input AND gate right after each of the XOR gates. One input of the gate will he the XOR output. The other input of the gate will be a signal with a constant delay (lower bounded by the 99% maximum PUF and XOR line delay. (Recall that the AND gate switches only if both of its inputs are ’1’.) The constant delay is generated by a line of buffers. The output of this line is fed into the second port of the AND gate. (Care shall be taken to equalize the routing to all the AND gates.) Now, the outputs of the AND gates will all switch simultaneously (or within a short time interval based on the AND gate process variations). In this way, the timing of output of the XOR gates shall be constant and (mostly) no-leaking information about the internal values before the XOR gates.

IX. Summary and Conclusions

In this paper, we investigated the reach of combined modeling and side channel attacks on electrical Strong PUFs architectures. We exemplified our attacks on XOR Arbiter PUFs and Lightweight PUFs, since they are considered two of the most secure electrical Strong PUF architectures [32], [34]. Previous works had found that these two PUFs could only be attacked successfully for up to five or six XORed single Arbiter PUFs, and for bitlengths of 64 bits or 128 bits [32], [34], by the state of the art in pure modeling attacks. XOR Arbiter PUFs or Lightweight PUFs with bitlenghts of 512 and eight single Arbiter PUFs had explicitly been suggested as both practical (i.e., sufficiently stable) and fully secure in earlier works [32], [34].

The two side channels we have suggested and examined in this work were power tracing of the arbiter element (i.e., the latch) in Arbiter PUFs and variants thereof, and marking different response patterns with corresponding timing signatures. These are the first power and timing side channels for PUFs reported in the literature. Both tell us the cumulative number of zeros and ones in the outputs of the k parallel Arbiter PUFs within an XOR Arbiter PUF or Lightweight Arbiter PUF structure. Taken by itself, this side channel info is almost worthless, since the attacker does not learn *which* of the single Arbiter PUF outputs is zero or one. As we showed in this paper, however, it is possible to adapt existing ML techniques in such a way that they can efficiently exploit this simple side channel information. This adaption turned out to be non-trivial, and constitutes one of the main contributions of this paper.

It very strongly boosts ML performance, and reduces the ML training complexity and CRP consumption on the above two PUFs from exponential [32], [34] to an empirically estimated low degree polynomial in this work. Without countermeasures against our side channels, the two above PUFs can hence no longer be regarded secure: Their practical instability increases exponentially in the number of XORs [32], [34], while the attack complexity (or security) rises only low-degree polynomial in this parameter.

Firstly, we attacked XOR Arbiter PUFs and Lightweight PUFs for up to 16 XORs and bitlengths of up to 512 on simulated side channel info and CRPs (see Table II). The use of simulated data poses no very strong restriction on our results: Earlier investigations had shown that simulated CRPs are very close to silicon CRPs, and had explicitly suggested that they can be used very well in any security analyses of Arbiter PUFs variants [34]. Furthermore, the error tolerance of our ML algorithms has been demonstrated in [32], [34], meaning that results obtained on error-free side channels should carry over well to the silicon case. Please note that the use of simulated data was an necessary step in order to conduct a large number of ML experiments, and to empirically estimate the complexity of our attacks on a sound basis.

Secondly, we carried out a full proof of concept for modeling and timing side channel attacks on FPGA implementations of the XOR Arbiter PUF and the Lightweight PUF (see Table III). They indeed confirmed that our results on simulated data carry over with little performance loss to the silicon case. The sizes and complexities we attacked successfully were 8 XORs and 12 XORs with 64, 128, 256, and 512 bits. One reason for choosing these specific sizes was that at CCS 2010 and IEEE T-IFS 2013 [32], [34], 8 XORs with bitlength 512 had been been suggested explicitly in theory as both practical (i.e., stable) and secure against pure modeling attacks. We stress that comparable sizes of the above two PUF types has never been implemented in silicon prior to our work. We implemented them for the first time, and subsequently tackled them successfully by our new attack method.

Finally, we discussed countermeasures against our side channels. They could and should be put in place in order to secure future Arbiter PUF generations against this method. Our countermeasure against the power side channel consists of using two symmetric, inverted output signals with two latches. This construction neutralizes and balances power consumption, regardless of PUF’s output. Interestingly, it can also be used to detect and stabilize output errors in Arbiter PUF variants, even though we did not work out this possibility in detail in this paper. We also discussed some countermeasure against our timing side channels, focusing on the construction of an isochronous hardware.

The PUF-attacks presented in this and other recent papers could be seen as a natural consolidation process in the PUF area, similar to the detailed investigations that classical cryptoprimitives and security systems have already undergone in the last decades. We believe that this interplay between attacks and countermeasures in the long term might well be beneficial for PUFs. Similar as in the case of classical security systems, it could eventually lead to improved PUF designs and implementations, which withstand all known attack forms at some point.

References

[1] Christopher M. Bishop, Nasser M. Nasrabadi: *Pattern recognition and machine learning*. Springer, New York, 2006.

[2] Christina Bruzska, Marc Fischlin, Heike Schröder, Stefan Katzenbeisser: *Physically Unclonable Functions in the Universal Composition Framework*. CRYPTO 2011.

[3] Jeroen Delvaux, Ingrid Verbauwhede: *Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise*. HOST 2013.

[4] Srinivas Devadas: *Physical unclonable functions and secure processors*. Invited talk, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009), September 2009.

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

[6] Blaise Gassend, Dwaine Clarke, Marten van Dijk, Srinivas Devadas: *Silicon physical random functions*. ACM Conference on Computer and Communications Security 2002: 148-160

[7] Blaise Gassend, Daihyun Lim, Dwaine Clarke, Marten v. Dijk, Srinivas Devadas: *Identification and authentication of integrated circuits.* Concurrency and Computation: Practice & Experience, pp. 1077 - 1098, Volume 16, Issue 11, September 2004.

[8] Jorge Guajardo, Sandeep S. Kumar, Geert Jan Schrijen, Pim Tuyls: *FPGA Intrinsic PUFs and Their Use for IP Protection*. CHES 2007: 63-80

[9] Clemens Helfmeier, Dmitry Nedospasov, Christian Boit, Jean-Pierre Seifert: *Cloning Physically Unclonable Functions*. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST'13), 2013.

[10] Gabriel Hospodar, Roel Maes, Ingrid Verbauwhede: Machine learning attacks on 65nm Arbiter PUFs: Accurate modeling poses strict bounds on usability. WIFS 2012: 37-42

[11] Khodjasteh, K., Sastrawan, J., Hayes, D., Green, T. J., Biercuk, M. J., and Viola, L: *Designing a practical high-fidelity long-time quantum memory*. Nature Communications, 4, 2013.

[12] J.-W. Lee, Daihyun Lim, Blaise Gassend, G. Edward Suh, Marten v. Dijk, and Srinivas Devadas. *A technique to build a secret key in integrated circuits with identification and authentication applications.* In Proceedings of the IEEE VLSI Circuits Symposium, June 2004.

[13] Daihyun Lim: *Extracting Secret Keys from Integrated Circuits*. MSc Thesis, MIT, 2004.

[14] Sandeep S. Kumar, Jorge Guajardo, Roel Maes, Geert Jan Schrijen, Pim Tuyls: *The Butterfly PUF: Protecting IP on every FPGA*. HOST 2008: 67-70

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

[16] Mehrdad Majzoobi, Farinaz Koushanfar, Miodrag Potkonjak: *Lightweight Secure PUFs.* IC-CAD 2008: 607-673.

[17] Mehrdad Majzoobi, Farinaz Koushanfar, Miodrag Potkonjak: *Testing techniques for hardware security.* In Proceedings of the International Test Conference (ITC), pages 1-10, 2008.

[18] 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.

[19] Dominik Merli, Dieter Schuster, Frederic Stumpf und Georg Sigl: *Side-Channel Analysis of PUFs and Fuzzy Extractors*. Conference on Trust and Trustworthy Computing (TRUST 2011). Lecture Notes in Computer Science, 2011, Volume 6740/2011, 33-47.

[20] Dominik Merli, Dieter Schuster, Frederic Stumpf, Georg Sigl: *Semi-invasive EM attack on FPGA RO PUFs and countermeasures*. ACM Workshop on Embedded Systems Security (WESS'11), 2011.

[21] Dmitry Nedospasov, Clemens Helfmeier, Jean-Pierre Seifert, Christian Boit: *Invasive PUF Analysis*. Fault Diagnonsis and Tolerance in Cryptography (FDTC'13), 2013.

[22] Erdinc Öztürk, Ghaith Hammouri, Berk Sunar: *Towards robust low cost authentication for pervasive devices.* In PerCom, pages 170-178. IEEE Computer Society, 2008.

[23] Ravikanth Pappu: *Physical One-Way Functions*. PhD Thesis, Massachusetts Institute of Technology, 2001.

[24] Ravikanth Pappu, Ben Recht, Jason Taylor, Neil Gershenfeld: *Physical One-Way Functions*, Science, vol. 297, pp. 2026-2030, 20 September 2002.

[25] M. Riedmiller, H. Braun: *A direct adaptive method for faster backpropagation learning: The RPROP algorithm*. IEEE international conference on neural networks, pp. 586–591, 1993.

[26] Ulrich Rührmair: *Oblivious Transfer based on Physical Unclonable Functions (Extended Abstract).* TRUST 2010.

[27] Ulrich Rührmair, Marten van Dijk: *Practical Security Analysis of PUF-based Two-Player Protocols.* CHES 2012.

[28] Ulrich Rührmair, Marten van Dijk: *On the Practical Use of Physical Unclonable Functions in Oblivious Transfer and Bit Commitment Protocols*. Journal of Cryptographic Engineering (JCEN), 2013.

[29] Ulrich Rührmair, Marten van Dijk: *PUFs in Security Protocols: Attack Models and Security Evaluations*. IEEE Symposium on Security and Privacy (Oakland'13), 2013.

[30] Ulrich Rührmair, Srinivas Devadas, Farinaz Koushanfar: *Security based on Physical Unclonability and Disorder*. In M. Tehranipoor and C. Wang (Editors): "Introduction to Hardware Security and Trust". Springer, 2011.

[31] Ulrich Rührmair, Christian Jaeger, Michael Algasiner: *An Attack on PUF-based Session Key Exchange, and a Hardware-based Countermeasure: Erasable PUFs.* Financial Cryptography and Data Security 2011.

[32] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, Jürgen Schmidhuber: *Modeling Attacks on Physical Unclonable Functions*. ACM Conference on Computer and Communications Security, 2010.

[33] Ulrich Rührmair, Jan Sölter, Frank Sehnke: *On the Foundations of Physical Unclonable Functions.* Cryptology e-Print Archive, June 2009.

[34] Ulrich Rührmair, Jan Sölter, Frank Sehnke, Xiaolin Xu, Ahmed Mahmoud, Vera Stoyanova, Gideon Dror, Jürgen Schmidhuber, Wayne Burleson, Srinivas Devadas: *PUF Modeling Attacks on Simulated and Silicon Data*. IEEE Transactions on Information Forensics and Security (IEEE T-IFS), 2013.

[35] G. Edward Suh, Srinivas Devadas: *Physical Unclonable Functions for Device Authentication and Secret Key Generation.* DAC 2007: 9-14

[36] Pim Tuyls, Geert Jan Schrijen, Boris Skoric, Jan van Geloven, Nynke Verhaegh, Rob Wolters *Read-Proof Hardware from Protective Coatings.* CHES 2006: 369-383