Policy Gradients for Cryptanalysis

Frank Sehnke, Christian Osendorfer, Jan Sölter, Jürgen Schmidhuber, and Ulrich Rührmair

Faculty of Computer Science, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
http://www6.in.tum.de/Main/WebHome

Abstract. Physical Unclonable Functions (PUFs) are an emerging, new cryptographic and security primitive. In this paper, we investigate to which extent the security of known PUFs can be challenged by a new machine learning technique named Policy Gradients with Parameter-based Exploration (PGPE). We apply PGPE to various PUF architectures and compare the results to other ML techniques obtained in earlier publications. Our findings show that PGPE have several advantages in PUF-cryptanalysis. Firstly, they merely require an internal parametric model of the PUF, which usually can be identified very easily for any circuit-based PUF. In opposition to that, several other ML methods such as SVMs or LR require (approximately) linearly separable or differentiable models, which are harder to develop. Secondly, we show by comparative experiments that PGPE outperforms the only other established ML method which works via a parametric model, namely Evolution Strategies, in the cryptanalysis of various PUFs.

1 Introduction

Physical One-Way Functions, Physical Random Functions and Physical Unclonable Functions are emerging recently as a powerful alternative to standard, mathematically-based cryptography and security. Eventhough the two former terms have been coined first, all three notions are often subsumed under the term Physical Unclonable Function or PUF [7].

It has been realized relatively early [6], however, that machine learning techniques are a natural and also a very powerful tool to challenge the security conditions of PUFs. In typical PUF applications, an adversary will be able to obtain a significant number of input bit vectors called challenges and the produced outputs called responses. The neccesary Challenge Response Pairs (CRPs)will be obtained either by eavesdropping on the communication protocol, or by gaining physical access to the PUF for a limited time period and measuring many CRPs himself. He can feed these CRPs into an ML algorithm. If successfully trained, the algorithm will predict the PUF responses with high probability, thereby breaking its security. For this very reason, machine learning methods at the moment constitue the most significant cryptanalytic attacks on PUFs.

This paper investigates to which extent the currently published electrical PUFs are susceptible to recent Policy Gradient (PG) methods. Our results show that several structures suggested as PUFs can be attacked well by a recently published PG method called Policy Gradients with Parameter-based Exploration (PGPE). The particular advantage of PGPE is that they merely require a parametric model of the attacked PUF, which is usually very easy to identify. PGPE is even faster and more reliable in breaking PUFs than population based heuristics (Evolution Strategies), which were successfully applied for cryptanalysis in another recent publication of our group [9]. The CRPs that we used in our experiments were generated by a linear additive delay model, as standard in the area.

Related Work Attacks on PUFs via machine learning algorithms have also been considered, for example, in [6, 4, 14]. [6] show that Support Vector Machines (SVMs) can successfully learn standard Arbiter PUFs. [4] prove that Feed-Forward Arbiter PUFs with one feed-forward loop can be learned by SVMs, too. Essentially the same result has been obtained independently by [14]. However, it can be shown that the approach of [4, 14] does not generalize to more than one feed-forward loop; two or more loops require new techniques. XOR Arbiter PUF and Lightweight PUF have, to the knowledge of the authors, only been cryptanalysed in [9].

Organization of the Paper The paper is organized as follows. In section **??** we define the investigated Arbiter PUF architectures. In section 3 we describe the population based heuristic used, namely Evolution Strategies

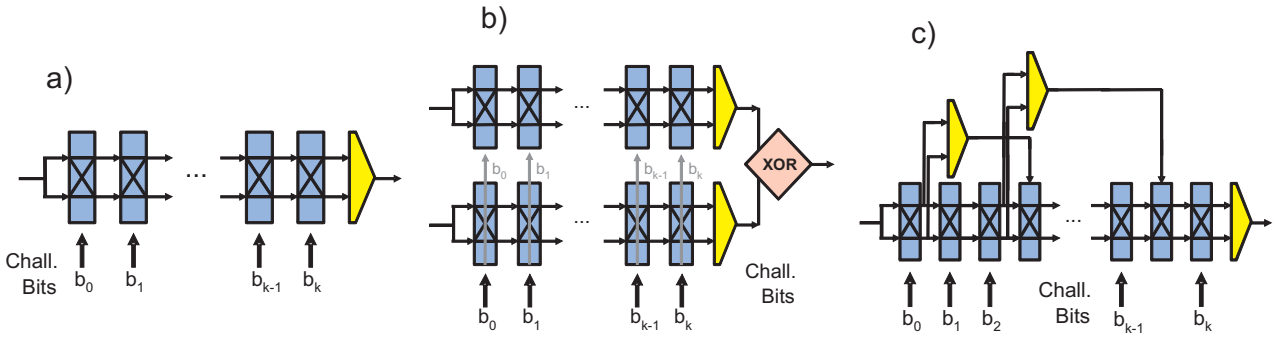

Fig. 1. Standard Arbiter PUF (left), XOR Arbiter PUF (middle) and Feed Forward Arbiter PUF (right) architecture scheme. The incoming bits to each stage decide if the signal propagates directly through the stage or if the signal is swoped. All ways the signal can take through the stage have slightly different run time properties due to small fabrication variances that are not controllable by the fabricator. The Arbiter at the end of the Standard Arbiter PUF outputs a bit depending on the signal coming first in the upper or the lower way. In the XOR Arbiter PUF several of the Standard Arbiter PUFs are the inputs to a XOR element. In FF Arbiter PUFs arbiters like the one at the end of the Standard Arbiter PUF are used to generate a bit depended on the run time differences after a certain stage to replace the normally incoming bit to a later stage.

(ES) and PGPE. Section 4 gives the results we obtained, in particular the obtained prediction rates plus the required CRPs and computation times for each ML method on each examined PUF. Section 5 summarizes the paper and discusses conclusions of our work.

2 Physical Unclonable Functions and Arbiter PUFs

Physical Unclonable Functions In a nutshell, a PUF is a physical system S with a unique, partly disordered fine structure that depends on uncontrollable manufacturing variations. The system can be exposed to external stimuli or "challenges". It reacts by returning so-called "responses", whose value depends on said manufacturing variations.

The special, security relevant features of a (Strong) PUF S are the following: (i) Due to the partly random finestructure of S, which should be beyond the control of its manufacturer, it must be impossible to fabricate a second physical system S' which has the same challenge-response behavior as S. (ii) Due to the complicated internal interactions of S, it must be impossible to devise a computer program that correctly predicts the response to a given challenge with high probability. This should hold even if many challenge-response pairs of S were known.

Together, the two conditions imply that the responses of S can be evaluated correctly only by someone who has got direct physical access to the single, unique system S. The validity of this assertion is essential for the security of all PUF-based protocols and schemes.

PUFs based on Runtime Delays in Integrated Circuits All electrical PUFs analyzed in this paper have some common characteristics: The state of some switches is configured by a challenge vector $\boldsymbol{C}$ (with the $i-th$ component encoding the state of the $i-th$ switch), leading to pairs of unique propagation pathes for an electric signal. The resulting propagation delay difference Δ between the pairs of pathes is then further transformed by an arbiter gate (respectivly combined arbiter and Xor gates) to a binary response t. Assuming that the overall propagation delay of a path is just the sum of the constant propagation delays of its constituent sub pathes, Gassend et al. established a parametric linear model for the propagation delay difference [2]. In a compact notation the model is given by

$$\Delta = \boldsymbol{w}^T \boldsymbol{\Phi} \tag{1}$$

Thereby the parameter vector $\boldsymbol{w}$ encodes the sub delays for all switches, whereas the feature vector $\boldsymbol{\Phi}$ is soley a function of the applied challenge vector $\boldsymbol{C}$. For the details, see [2] and [6].

As shown in [6], the set of all possible linear propagation difference delay models (eqn 1) covers the characteristic of real PUF instances sufficient well, such that each PUF instance can be assigned a model

instance with its response prediction error in the range of the PUFs real-world stability. Therefore in this paper algorithms are presented which determine the suitable parameters $\boldsymbol{w}$ provided that an adequate solution is contained in the set of linear propagation delay models. That is, the algorithms are evaluated by applying them to data generated by the linear model itself with the sub delays drawn from a Gaussian distribution [13].

3 Employed Machine Learning Methods

Evolution Strategies Evolution Strategies (ES) [10] belong to a class of ML techniques called Evolutionary Algorithms. They are inspired by the biological evolution of a population of individuals under certain environmental conditions. The population repeatedly undergoes the evolutionary steps of evaluation, environment selection, partner selection, recombination and mutation. In this process, individuals evolve which fit increasingly well a previously specified target.

In our case, an individual is given by a concrete instantiation of the runtime delays in a PUF (or by the vector $\boldsymbol{w}$ from equation (1)). The environmental fitness is determined by how well this individual (re-)produces the correct CRPs of the target PUF as output. The outputs of the individual are computed by a linear additive delay model from its subdelays (or from $\boldsymbol{w}$), and are compared to several known outputs of the target PUF structure. The best individuals are selected. In the following recombination step, the remaining individuals mutually exchange part of their 'genome'/their individual subdelays, in order to form descendants. In the final mutation step, the subdelays are varied randomly, and the process starts anew.

ES are especially suited for this task since in problems with causal search spaces, ES are known to be among the most effective methods. The use of special mutation operators as introduced by Rechenberg and Schwefel [8, 11] make them self-adapting to the properties of the search space.

We used a standard implementation of ES with the ES standard meta-parameters [1]: Population size of (6,36), comma-best-selection, and a global mutation operator with $\tau = \frac{1}{\sqrt{(n)}}$.

Policy Gradients with Parameter-based Exploration In what follows, we briefly summarize [12], outlining the derivation that leads to PGPE. We give a short summary of the algorithm as far as it is needed for the rest of the paper. Consider an agent interacting with an environment. Denote the state of the environment at time t as s_t and the action at time t as a_t. Because we are interested in continuous state and action spaces (usually required for control tasks), we represent both a_t and s_t with real valued vectors. We assume that the environment is Markovian, i.e. that the current state-action pair defines a probability distribution over the possible next states $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. We further assume that the actions depend stochastically on the current state and some real valued vector θ of agent parameters: $a_t \sim p(a_t|s_t, \theta)$. Lastly, we assume that each state-action pair produces a scalar reward $r_t(a_t, s_t)$. We refer to a length T sequence of state-action pairs produced by an agent as a history $h = [s_{1:T}, a_{1:T}]$.

Given the above formulation we can associate a cumulative reward r with each history h by summing over the rewards at each time step: $r(h) = \sum_{t=1}^{T} r_t$. In this setting, the goal of reinforcement learning is to find the parameters θ that maximize the agent's expected reward

$$J(\theta) = \int_H p(h|\theta) r(h) dh \tag{2}$$

An obvious way to maximize $J(\theta)$ is to find $\nabla_\theta J$ and use it to carry out gradient ascent. Noting that the reward for a particular history is independent of θ, and using the standard identity $\nabla_x y(x) = y(x) \nabla_x \log y(x)$, we can write

$$\nabla_\theta J(\theta) = \int_H \nabla_\theta p(h|\theta) r(h) dh = \int_H p(h|\theta) \nabla_\theta \log p(h|\theta) r(h) dh \tag{3}$$

PGPE addresses the variance problem by replacing the probabilistic policy with a probability distribution over the parameters θ, where ρ are the parameters determining the distribution over θ. The advantage of this approach is that the actions are deterministic, and an entire history can therefore be generated from a single parameter sample. This reduction in samples-per-history is what reduces the variance in the gradient estimate. As an added benefit the parameter gradient is estimated by direct parameter perturbations, without

having to backpropagate any derivatives, which allows the use of non-differentiable controllers or modells. The expected reward with a given ρ is

$$J(\rho) = \int_{\Theta} \int_{H} p(h, \theta|\rho) r(h) dh d\theta. \tag{4}$$

Noting that h is conditionally independent of ρ given θ, we have $p(h, \theta|\rho) = p(h|\theta)p(\theta|\rho)$ and therefore $\nabla_\rho \log p(h, \theta|\rho) = \nabla_\rho \log p(\theta|\rho)$. Substituting this into Eq. (4) yields Eq. (5) under the notion of several conditionally independencies (given the details from [12]).

$$\nabla_\rho J(\rho) = \int_{\Theta} \int_{H} p(h|\theta) p(\theta|\rho) \nabla_\rho \log p(\theta|\rho) r(h) dh d\theta \tag{5}$$

where $p(h|\theta)$ is the probability distribution over the parameters θ and ρ are the parameters determining the distribution over θ. Clearly, integrating over the entire space of histories and parameters is unfeasible, and we therefore resort to sampling methods. This is done by first choosing θ from $p(\theta|\rho)$, then running the agent to generate h from $p(h|\theta)$:

$$\nabla_\rho J(\rho) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\rho \log p(\theta|\rho) r(h^n) \tag{6}$$

We assume that ρ consists of a set of means $\{\mu_i\}$ and standard deviations $\{\sigma_i\}$ that determine an independent normal distribution for each parameter θ_i in θ. (In this case this assumption is especially useful because the fabrication variances of the Arbiter PUFs are around a known μ with an also known σ and are very well normal distributed and the delays in the PUF architecture are independent.)

Some rearrangement gives the following forms for the derivative of $\log p(\theta|\rho)$ with respect to μ_i and σ_i:

$$\nabla_{\mu_i} \log p(\theta|\rho) = \frac{(\theta_i - \mu_i)}{\sigma_i^2} \qquad \nabla_{\sigma_i} \log p(\theta|\rho) = \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3}, \tag{7}$$

which can then be substituted into (6) to approximate the μ and σ gradients. Note the similarity to REINFORCE [15], but in contrast to REINFORCE θ resembles the parameters of the modell not the taken action. This enables us to use this mehtod on models that are non-differentiable like the FF-Arbiter PUFs.

We used the standard implementation of PGPE with the PGPE standard meta-parameters [12]: 2-Sample Symetric Sampling, starting standard diviation for exploration as the standard diviation assumed for the PUFs and stepsizes of 0.2 and 0.1 for the parameter and the sigma update. We also applied the usual reward normalization for PGPE.

4 Results

We will now discuss the results that we achieved in the application of the above machine learning techniques to the currently known electrical PUFs. If not stated differently, as the training data underlying the experiments, we used a set of 50.000 CRPs with random subsets of 2.000 CRPs for the evaluation step of the individuals. These CRPs where generated on the basis of a linear additive delay model. The subdelays in the stages were drawn according to a uniform distribution with parameters motivated by the standard fabrication delays that may occur in such a structure. Please note that since ES and PGPE are probabilistic methods, additional evaluations and also additional runs can be expected to yet further improve the prediction accuracys presented in this section. Also more suitable meta-parameters like bigger population sizes or smaller τ for ES or smaller step sizes for PGPE can optimize the performance further.

4.1 Standard Arbiter PUFs

Evolution Strategies Figure 2 shows the average of 10 runs on each standard arbiter PUF that have been conducted. As shown, in all cases we have been able to successfully learn the PUFs nearly perfect (99.26% for 64 bit) in only 10,800 evaluations. In other words, the standard Arbiter PUF can be broken well by ES, and also by several other ML methods ([2, 9]).

Table 1 shows the evaluations needed to achieve an average prediction rate of 10 % and 5 %. The need of evaluations seem to grow only linearly with the number of bits.

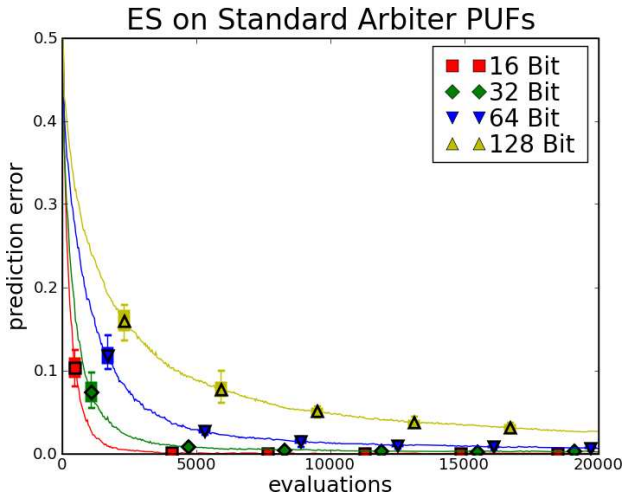

Fig. 2. Results of 10 runs per experiment for different lengths of Standard Arbiter PUFs with ES

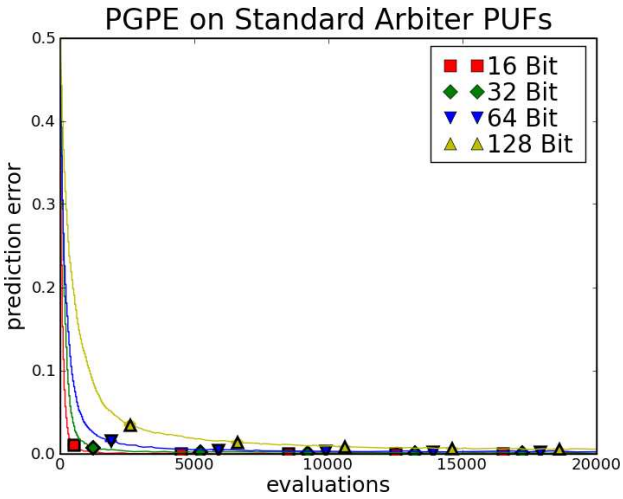

Fig. 3. Results of 10 runs per experiment for different lengths of Standard Arbiter PUFs with PGPE

ES on Standard Arbiter PUFs:

	10%		5 %	
Bit	E	E/Bit	E	E/Bit
16	446	27.88	720	45.00
24	680	28.33	1202	50.08
32	878	27.44	1530	47.81
48	1264	26.33	2387	49.73
64	1879	29.36	3589	56.08

Table 1. The evaluations needed to achieve an average prediction rate of 10 % and 5 % with ES. E marks the columns with the average evaluations needed, while E/Bit marks the columns that shows the evaluations needed per number of bits.

PGPE on Standard Arbiter PUFs:

	10%		5 %	
Bit	E	E/Bit	E	E/Bit
16	118	7.38	190	11.88
24	171	7.13	280	11.67
32	219	6.84	384	12.00
48	357	7.44	605	12.60
64	467	7.30	834	13.03

Table 2. The evaluations needed to achieve an average prediction rate of 10 % and 5 % with PGPE. E marks the columns with the average evaluations needed, while E/Bit marks the columns that shows the evaluations needed per number of bits.

Policy Gradients with Parameter-based Exploration Figure 3 shows the average of 10 runs on each standard arbiter PUF that have been conducted. In all cases we have been able to successfully learn the PUFs nearly perfect (99.84% for 64 bit) in only 10800 evaluations. In other words, Standard Arbiter PUF can also be broken by PGPE. This confirms earlier results obtained by other groups [6, 2–4, 13].

Table 2 shows the evaluations needed to achieve an average prediction rate of 10 % and 5 %. The need of evaluations seem to grow only linearly with the number of bits.

When comparing PGPE and ES, we observe that PGPE after the same number of generations, PGPE achieves a remaining prediction error that is smaller by a factor of 5 than ES. Further, PGPE performs computationally about 4 times faster on this basic type of Arbiter PUF.

4.2 XOR Arbiter PUF

In contrast to the other experiments we used random subsets of 8.000 CRPs for the evaluation step of the individuals for all XOR Arbiter PUF experiments. We assumed that all XOR'ed circuits have equal input signal vectors.

Model ToDo: HERE we should write about the model that we applied for the XOR, and why the application of PGPE makes is easy to devise a model.

Evolution Strategies Figures 4 to 6 show the best of a total of 10 runs on each XOR-Arbiter PUF that have been conducted. Table 3 shows the needed evaluations to achieve a prediction rate of 10% and the fraction

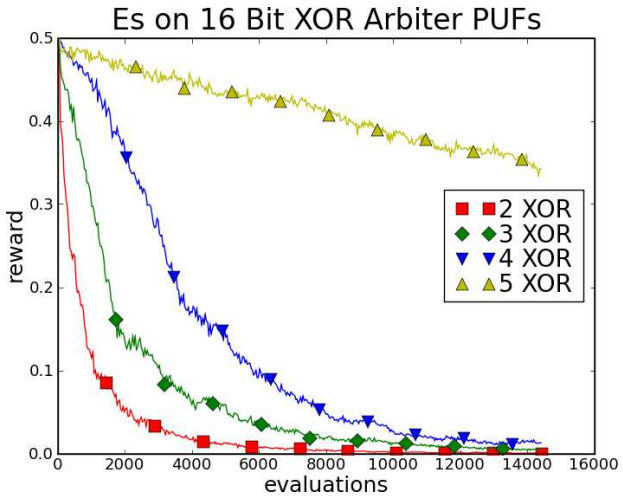

Fig. 4. The best of 10 runs on each XOR-Arbiter PUF architecture with an 16 bit input vector.

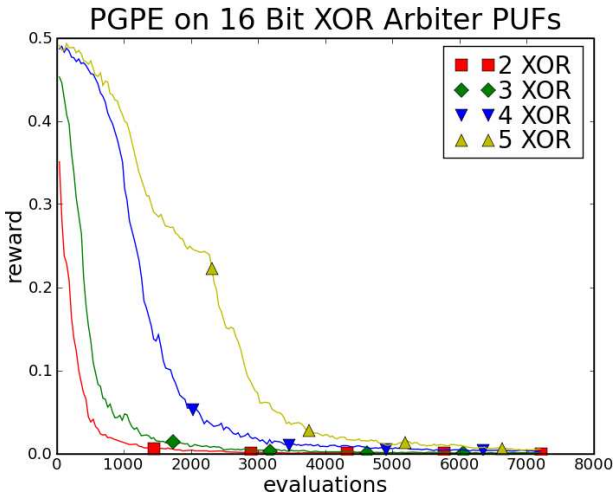

Fig. 5. The best of 10 runs on each XOR-Arbiter PUF architecture with an 16 bit input vector.

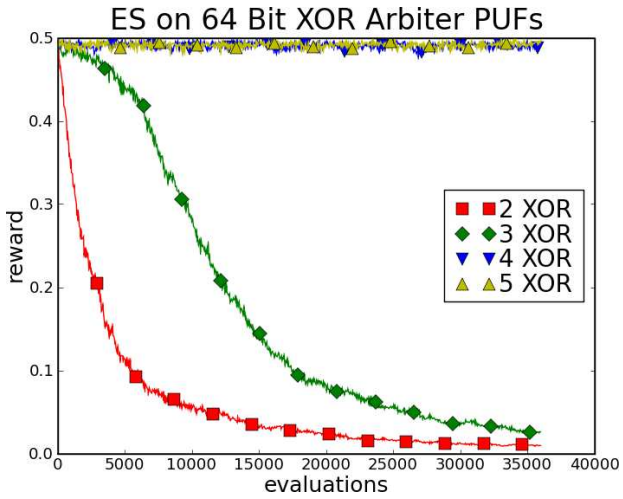

Fig. 6. The best of 10 runs on each XOR-Arbiter PUF architecture with an 16 bit input vector.

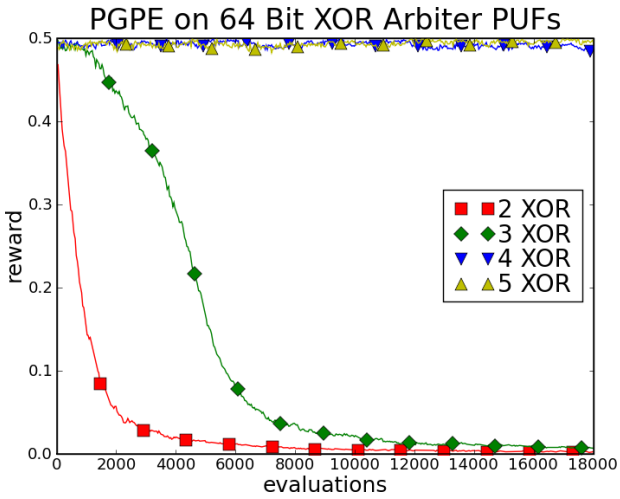

Fig. 7. The best of 10 runs on each XOR-Arbiter PUF architecture with an 16 bit input vector.

of runs that have achieved this prediction rate in the given maximal number of evaluation steps. Clearly the number of evaluations needed grows more than linearly. If the needed evaluations grow exponential or polynomial with respect to the number of XOR-inputs is hard to say. Also the fraction of runs that succeed in predicting the PUF with a suitable rate drops drastically with the number of XOR-inputs. However, as shown, in all cases up to 3 XOR-inputs we have been able to successfully learn the PUFs with rates better than 10%. In this sense, our experiments show that the XOR-Arbiter can in principal be broken by ES up to at least 3 XOR-inputs.

Policy Gradients with Parameter-based Exploration Figures 5 to 7 show the best of a total of 10 runs on each XOR-Arbiter PUF that have been conducted. Table 4 shows the necessary evaluations to achieve a prediction rate of 10% and the fraction of runs that have achieved this prediction rate in the given maximal number of evaluation steps. Clearly the required number of evaluations grows also more than linearly for PGPE. Whether the necessary evaluations grow exponential or polynomial with respect to the number of XOR-inputs is again hard to say. Also the fraction of runs that succeed in predicting the PUF with a suitable rate drops drastically with the number of XOR-inputs, though it drops less drastically than for ES. However,

XOR	Gen 16 Bit	Gen 64 Bit	Rate 16 Bit	Rate 64 Bit
2	30.0	153.0	100%	100%
3	84.2	485.0	90%	50%
4	161.0	-	30%	0%
5	-	-	0%	0%

Table 3. The number of evaluations needed to achieve a prediction rate of 10% and the rate of runs that achieved this prediction rate in the given maximal number of evaluations. Gen marks the columns with the evaluations needed, while Rate marks the columns that show the rate of successful runs.

XOR	Gen 16 Bit	Gen 64 Bit	Rate 16 Bit	Rate 64 Bit
2	30.0	153.0	100%	100%
3	84.2	485.0	90%	50%
4	161.0	-	30%	0%
5	-	-	0%	0%

Table 4. The number of evaluations needed to achieve a prediction rate of 10% and the rate of runs that achieved this prediction rate in the given maximal number of evaluations. Gen marks the columns with the evaluations needed, while Rate marks the columns that show the rate of successful runs.

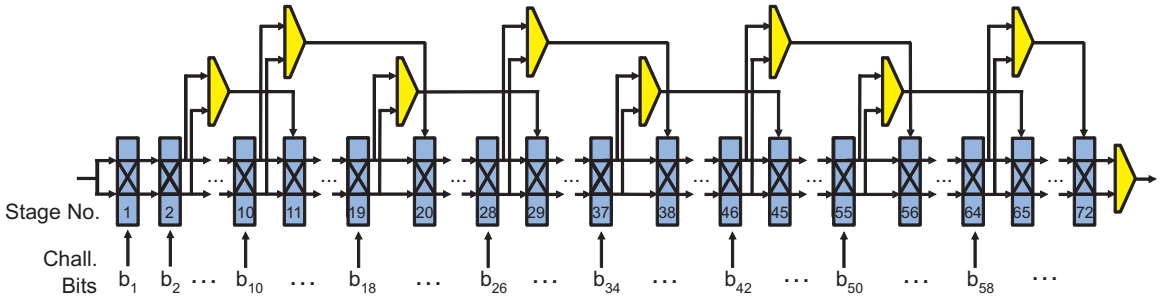

Fig. 8. The architecture of the FF-Arbiter PUFs that we employed in our ML experiments.

also PGPE was able in all cases up to 3 XOR-inputs to successfully learn the PUFs with rates better than 10%.

An interesting related question is how the stability of a real XOR-Arbiter behaves with increasing numbers of XOR inputs. The overall stability of the XOR PUF is the statistically combined stability of the single Standard Arbiter PUFs that are used as inputs to the XOR stage. For example, assuming that a single standard arbiter PUF has a stability of 98% over a temperature range of 45¡C, an XOR-Arbiter PUF with 5 XOR-inputs would only have an overall stability of 90.4% over the same temperature range.

4.3 Feed Forward Arbiter PUFs

Feed Forward Arbiter PUFs (FF Arb PUFs) are the most important type of PUF for this paper. Their models are, in general, not differentiable, and are therefore not prone to supervised learning and to standard PG methods. In [9], we showed that FF Arb PUFs are prone to attacks based on Evolutionary Algorithms. In this section, we show that PGPE is a better alternative to ES in breaking this PUF. The architecture of the chosen PUF structures are shown in Figure 8.

Model ToDo: Here we should write which model we applied, and why the application of PGPE makes it so easy to set up a model.

Evolution Strategies Figure 9 shows the best of a total of 10 runs on each FF-Arbiter PUF that have been conducted with ES. As shown, in all cases we have been able to successfully learn the PUFs with rates better than 97%. Please note that our accuracy is significantly better than the stability of an in-silicon FF-Arbiter with 7 FF-loops while undergoing a temperature change of 45C, which is only 90.16% [5]. In this sense, the experiments show that the FF-Arbiter are well susceptible to attacks by ES.

In the following we repeat the complexity estimates of the FF-Arbiter PUF from [9] with respect to the needed CRPs with growing number of bits. We chose architectures with 6 FF stages that are equally distributed over the PUF, and overlap in the same manner as shown in figure 8. The results are shown in Figure 13. Every data point resembles the average of 10 runs in prediction difference. Prediction difference here means the difference between the quality on the test and the training set. The given number of CRPs

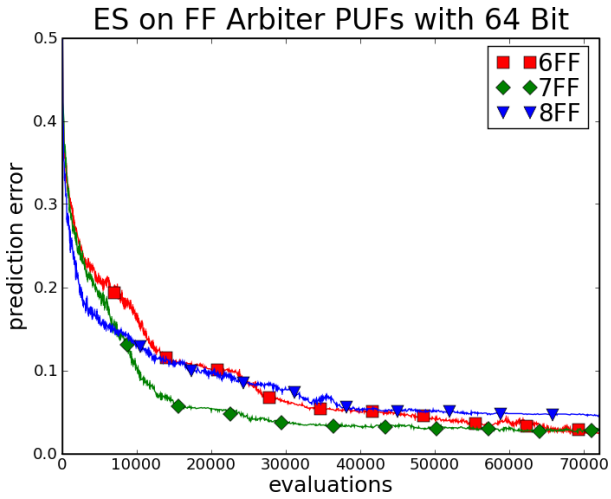

Fig. 9. The best of 40 runs on each FF-Arbiter PUF architecture with ES.

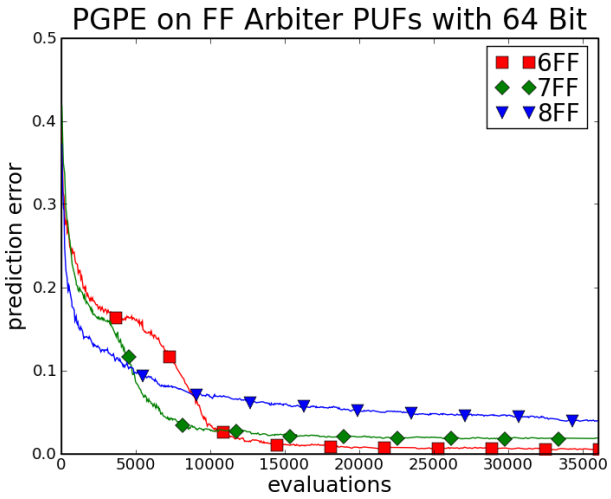

Fig. 10. The best of 10 runs on each FF-Arbiter PUF architecture with PGPE.

		10%		5%	Result	
FF	E	E/FF	E	E/FF	Best	Aver.
6	5832	972	9756	1626		
7	6528	933	11844	1692		
8	12544	1568	23112	2889		

Table 5. The evaluations needed to achieve an prediction rate of 10% and 5%. G marks the columns with the average evaluations needed, while G/Bit marks the columns that shows the evaluations needed per number of bits and G/St marks the columns that show the evaluations needed per number of stages.

		10%		5%	Result	
FF	E	E/FF	E	E/FF	Best	Aver.
6	1580	263	2870	478		
7	2210	316	3440	491		
8	4260	533	8120	1015		

Table 6. The evaluations needed to achieve an prediction rate of 10% and 5%. G marks the columns with the average evaluations needed, while G/Bit marks the columns that shows the evaluations needed per number of bits and G/St marks the columns that show the evaluations needed per number of stages.

were divided in half for these two sets. To reduce the noise that is still present with 10 runs per data point, we fitted a hyperbola on the data points (see figure 13. Table 9 shows the CRPs needed, estimated by the hyperbola fit, to reach a prediction error less than 10% and 5%. The results suggest a linear growth. For the complexity experiments of the FF-Arbiter PUF with respect to the needed CRPs with growing number of FF stages, we chose also architectures with 32 bit. The FF stages are again equally distributed over the PUF and overlap in the same manner like shown in figure 8. The results are shown in Figure 14. Every data point resembles the average of 10 runs in prediction difference. The given number of CRPs were again divided in half for the test and training sets. We also fitted a hyperbola on the data points. Table 10 shows the CRPs needed, estimated by the hyperbola fit, to reach an prediction difference less than 10% and 5%. The results suggest a less than linear growth. The growth in need of evaluations is also of interest. Figure 11 shows the best of a total of 10 runs on each FF-arbiter PUF that have been conducted. Table 8 shows the evaluations needed to achieve an average prediction rate of 10% and 5%. The need of evaluations seem again to grow only linearly with the number of bits. Because the computation time for simulating an Arbiter PUF grows also linearly with the number of stages, the complexity in computational time would be $O(n^2)$.

Policy Gradients with Parameter-based Exploration Figure 10 shows the best of a total of 10 runs on each FF-Arbiter PUF that have been conducted with ES. In all cases we have been able to successfully learn the PUFs with rates better than 98% with PGPE. The experiments show that the FF-Arbiter can be fully broken also by PGPE.

As there is no reason that PGPE would need more CRPs or respectively would grow in the need of CRPs more than ES we restrict our experiments in showing that with the same number of CRPs PGPE produces on the hardest PUFs investigated equaly or better results.

TODO: 32Bit8FF with different number of CRPs

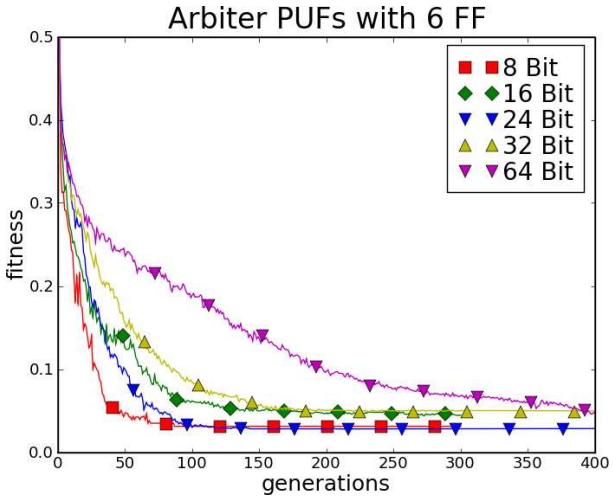

Fig. 11. The best of 10 runs on each FF-Arbiter PUF architecture.

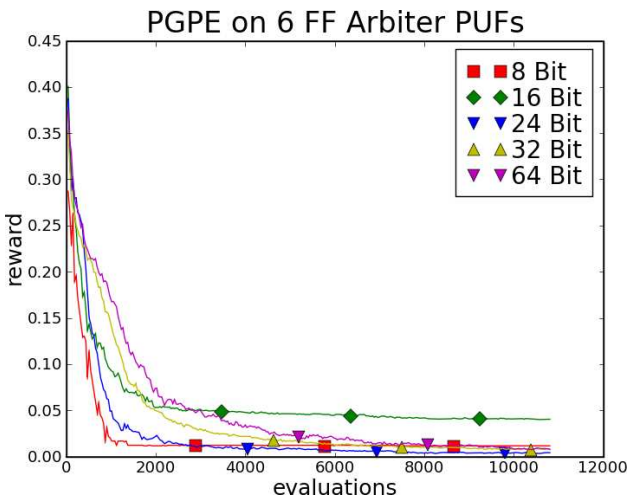

Fig. 12. The best of 10 runs on each FF-Arbiter PUF architecture.

	10%			5%		
Bit	G	G/Bit	G/St	G	G/Bit	G/St
8	28.7	3.59	2.05	43.3	5.41	3.09
16	62.2	3.89	2.83	139.0	8.69	6.32
24	47.6	1.98	1.59	74.4	3.10	2.48
32	84.0	2.63	2.21	218.0	6.81	5.74
64	196.0	3.06	2.80	393.0	6.14	5.61

Table 7. The evaluations needed to achieve an prediction rate of 10% and 5%. G marks the columns with the average evaluations needed, while G/Bit marks the columns that shows the evaluations needed per number of bits and G/St marks the columns that show the evaluations needed per number of stages.

	10%			5%		
Bit	G	G/Bit	G/St	G	G/Bit	G/St
8	28.7	3.59	2.05	43.3	5.41	3.09
16	62.2	3.89	2.83	139.0	8.69	6.32
24	47.6	1.98	1.59	74.4	3.10	2.48
32	84.0	2.63	2.21	218.0	6.81	5.74
64	196.0	3.06	2.80	393.0	6.14	5.61

Table 8. The evaluations needed to achieve an prediction rate of 10% and 5%. G marks the columns with the average evaluations needed, while G/Bit marks the columns that shows the evaluations needed per number of bits and G/St marks the columns that show the evaluations needed per number of stages.

5 Discussion and Conclusion

We showed that the recently published Policy Gradient method PGPE can be applied well to the cryptanalysis of various PUFs. It is easy to apply to all current and, with all likelihood, also to all future circuit based PUF implementations, since it merely required a parametric internal model of the PUF. We also showed that PGPE significantly outperforms Evolution Strategies in the cryptanalysis of PUFs. Due to its easy applicability, and as no hard-to-optimize differentiable or linearly separable models need to be set up, PGPEs have a high potential to become a standard benchmark method for the security of circuit based PUF implementations. ML curves obtained by PGPE on small PUF instances can help us to judge and compare the security of various PUF implementations.

References

1. T. Bäck: *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford University Press, USA, 1996.
2. Blaise Gassend, Daihyun Lim, Dwaine Clarke, Marten van Dijk, Srinivas Devadas: *Identification and authentication of integrated circuits.* Concurrency and Computation: Practice & Experience, pp. 1077 - 1098, Volume 16, Issue 11, September 2004.
3. Lee, ?.: *Please fill in teh actual publication.* ???: ???.
4. M. Majzoobi, F. Koushanfar, M. Potkonjak: *Lightweight Secure PUFs.* IC-CAD 2008: 607-673.
5. D. Lim: *Extracting Secret Keys from Integrated Circuits.* MIT, 2004
6. D. Lim: *Extracting Secret Keys from Integrated Circuits.* MSc Thesis, MIT, 2004.

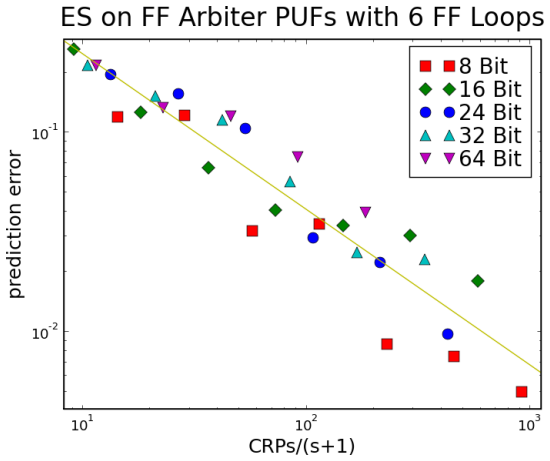

Fig. 13. Results of 10 runs per data point for different lengths of FF Arbiter PUFs and the hyperbola fits.

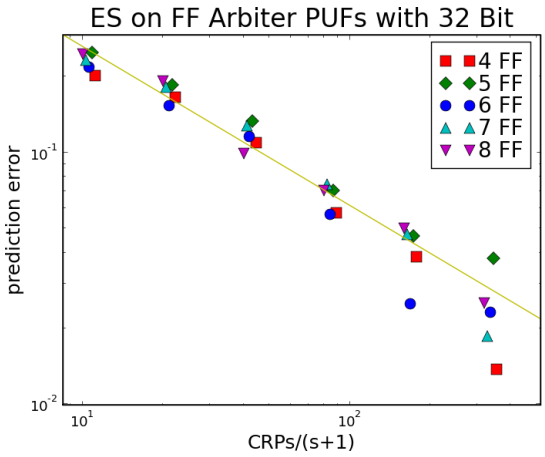

Fig. 14. Results of 10 runs per data point for different numbers of FF stages and the hyperbola fits.

	10%			5%		
Bit	CRP	C/Bit	C/St	CRP	C/Bit	C/St
8	447	55.88	31.93	940	117.50	67.14
16	872	54.50	39.64	1970	123.13	89.55
24	1150	47.92	38.33	2580	107.50	86.00
32	1710	53.44	45.00	3810	119.06	100.26

Table 9. The number of CRPs needed to achieve an average prediction error of 10% and 5%. CRP marks the columns with the CRPs needed, while C/Bit marks the columns that show the CRPs needed per number of bits and C/St marks the colums that show the CRPs needed per number of stages.

	10%			5%		
FF	CRP	C/FF	C/St	CRP	C/FF	C/St
2	953	476.50	28.03	2140	1070.00	62.94
4	1790	447.50	49.72	4010	1002.50	111.39
6	1750	291.67	46.05	3860	643.33	101.58
8	2300	287.50	57.50	5200	650.00	130.00

Table 10. The number of CRPs needed to achieve an average prediction error of 10% and 5%. CRP marks the columns with the CRPs needed, while C/FF marks the columns that show the CRPs needed per number of FF stages and C/St marks the colums that show the CRPs needed per number of stages.

7. R. Pappu, B. Recht, J. Taylor, N. Gershenfeld, *Physical One-Way Functions*, Science, vol. 297, pp. 2026-2030, 20 September 2002.
8. I. Rechenberg: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Fromman-Holzboog, Stuttgart, Germany, 1973.
9. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Stoyanova, V., Schmidhuber, J.: *Machine Learning Attacks on Physical Unclonable Functions.* Not published yet, 2010.
10. H.P. Schwefel: *Evolution and optimum seeking.* Wiley, New York, 1995.
11. H.-P. Schwefel: *Numerical Optimization of Computer Models.* John Wiley and Sons, LTD, 1981.
12. Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., and Schmidhuber, J.: *Parameter-exploring policy gradients.* Neural Networks, Special Issue, December 2009.
13. M. Majzoobi, F. Koushanfar, and M. Potkonjak: *Testing techniques for hardware security.* ITC, 2008
14. Vera Stoyanova: *Machine Learning and Physical Unclonable Functions.* MSc-Thesis, Technische Universitt Mnchen, 2008.
15. R.J. Williams: *Simple statistical gradient-following algorithms for connectionist reinforcement learning.*Machine Learning, 8:229–256, 1992.