Policy Gradients for Cryptanalysis

Frank Sehnke[1], Christian Osendorfer[1], Jan Sölter[2],
Jürgen Schmidhuber[3,4], and Ulrich Rührmair[1]

[1]Faculty of Computer Science, Technische Universität München, Germany
[2]Faculty of Biology, Freie Universität Berlin, Germany
[3]Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, Lugano, Switzerland
[4]Faculty of Computer Science, Universitá della Svizzera italiana, Lugano, Switzerland

Abstract. So-called Physical Unclonable Functions are an emerging, new cryptographic and security primitive. They can potentially replace secret binary keys in vulnerable hardware systems and have other security advantages. In this paper, we deal with the cryptanalysis of this new primitive by use of machine learning methods. In particular, we investigate to what extent the security of circuit-based PUFs can be challenged by a new machine learning technique named Policy Gradients with Parameter-based Exploration. Our findings show that this technique has several important advantages in cryptanalysis of Physical Unclonable Functions compared to other machine learning fields and to other policy gradient methods.

1 Introduction

Background on Physical Unclonable Functions. Physical Unclonable Functions (PUFs) are emerging recently as a powerful alternative to standard, mathematically based cryptography and security [1], [2]. 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.

Two typical applications of (Strong) PUFs are identification and key exchange scenarios. In these settings, PUFs have two advantages: (i) They avoid the storage of secret binary keys in vulnerable hardware systems, from which the keys can potentially be extracted by invasive attacks or viruses. (ii) They avoid the usual, unproven number theoretic assumptions that plague mathematical cryptography (albeit they rest on other assumptions). In other words, secure PUFs can create a new, advantageous form of cryptography in several application scenarios [1],[3],[4].

Machine Learning Attacks on PUFs. It has been realized relatively early in the history of PUFs [5] that Machine Learning (ML) techniques are a natural and also a very powerful tool to challenge the security of strong PUFs. In typical PUF applications, an adversary will be able to obtain a significant number

of challenges and corresponding responses (so-called challenge-response-pairs or CRPs) of the PUF. He can obtain the CRPs either by eavesdropping on communication protocols, or by gaining physical access to the PUF for a limited time period and measuring many CRPs himself. He can then feed these CRPs into an ML algorithm, interpreting the challenges as input and the responses as output of an unknown, to-be-learned function. If successfully trained, the algorithm will later predict the PUF's responses with high probability, thereby breaking the security of the PUF, and of all protocols derived from it.

Our Contributions. This paper investigates to which extent the currently published electrical PUFs are susceptible to recent Policy Gradient (PG) methods. We show that small or medium size instances of almost all candidates of electrical Strong PUFs can be attacked well by a recent PG method called Policy Gradients with Parameter-based Exploration (PGPE) [6]. Our investigations show that PGPE is a particularly general and an efficient ML method for the cryptanalysis of electrical Strong PUFs. First of all, they merely require a parametric model of the attacked PUF, which is usually easy to identify. Contrary to that, the ML methods from the Reinforcement Learning (RL) or Supervised Learning domain that were applied to PUF cryptanalysis so far [7] all required differentiable models. Such models are much harder to find, and sometimes may not even exist.

Secondly, PGPE is faster and more reliable in attacking PUFs than population based heuristics (Evolution Strategies (ES) [8]), which were applied for cryptanalysis in a recent other publication of our group [7]. ES is the only other known ML strategy capable of attacking PUFs via a merely parametric model.

Organization of the Paper. The paper is organized as follows. In section 2 we define the investigated Arbiter PUF architectures. In section 3 we describe the population based heuristic used, namely Evolution Strategies (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

As mentioned briefly in the introduction, a PUF is a physical system S with a unique, partly disordered fine structure that depends on uncontrollable manufacturing variations. The special security features of a (Strong) PUF S are the following: (i) Due to the partly random fine-structure 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 (CRPs) of S were known.

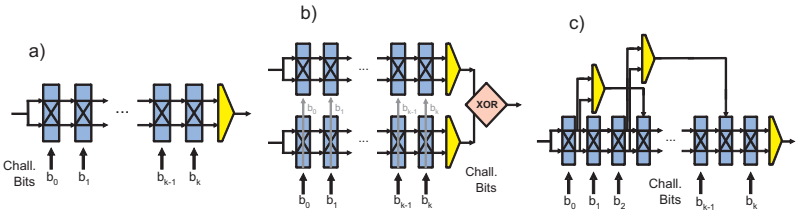

Fig. 1. Illustration of the architectures of a Standard Arbiter PUF (a), XOR Arbiter PUF (b) and Feed Forward Arbiter PUF (c). The challenge bits b_i at each stage decide if the two incoming signals propagate in parallel through the stage, or if their paths are crossed. All signal paths have slightly different run time properties due to uncontrollable, small fabrication variations. An arbiter element depicted as yellow at the end of the Standard Arbiter PUF (a) decides which of the two signals arrived first, and correspondingly outputs 0 or 1. In an XOR Arbiter PUF (b), the output of several Standard Arbiter PUFs is XORed. In FF Arbiter PUFs, signals at earlier stages of the circuit are fed into an arbiter element, whose output is applied as external bit at later stages of the circuit.

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. But exactly this assertion can be challenged by ML techniques: A successfully trained ML algorithm can imitate a PUF's responses numerically, and can be copied and distributed at will.

All electrical PUFs analyzed in this paper have some common characteristics (see figure 1): 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 paths for an electric signal. The resulting propagation delay difference Δ between the pairs of paths is then further transformed by an arbiter gate (respectively 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 paths, Gassend et al. established a parametric linear model for the propagation delay difference [9]. In a compact notation the model is given by

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

where $\boldsymbol{w}$ and $\boldsymbol{\Phi}$ are of dimension $k+1$. The parameter vector $\boldsymbol{w}$ encodes the delays for the subcomponents in the stages, whereas the feature vector $\boldsymbol{\Phi}$ is solely a function of the applied $k-$bit challenge C [5] [10] [11].

As shown in [5], the set of all possible linear propagation difference delay models (eq. 1) covers the characteristic of real PUF instances sufficiently 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 [11]. The detailed models for each PUF variant are explained in the subsections of section 4.

3 Employed Machine Learning Methods

Evolution Strategies. We chose Evolution Strategies as a benchmark for PGPE because they have been used in earlier publications by our group [7]. Up to now, they were the only ML method that was, at least in principle, applicable to all known electrical PUFs, since they merely required a parametric model of the PUF.

To attack PUFs with ES, an individual in the ES-population 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. We used a standard implementation of ES with the ES standard meta-parameters [12]: 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 [6] and [13], 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. We assume that every executed episode or role out produces a scalar reward r. 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 explores the search space by a probability distribution over the parameters θ, where ρ are the parameters determining the distribution over θ. The parameter gradient is therefore estimated by direct parameter perturbations, without having to backpropagate any derivatives, which allows the use of non-differentiable controllers or models (unlike standard PG methods). The expected reward with a given ρ is

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

Under the notion of several conditional independencies (given the details from [6]) and by refering to sampling methods, we get:

$$\nabla_\rho J(\rho) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\rho \log p(\theta|\rho) r(h^n) \tag{5}$$

Sampling is done by first choosing θ from $p(\theta|\rho)$, then running the agent to generate h from $p(h|\theta)$. 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:

$$\nabla_{\mu_i} \log p(\theta|\rho) = \frac{(\theta_i - \mu_i)}{\sigma_i^2} \qquad \nabla_{\sigma_i} \log p(\theta|\rho) = \frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3}, \tag{6}$$

which can then be substituted into (5) to approximate the μ and σ gradients.

We used the standard implementation of PGPE with the PGPE standard meta-parameters [6]: 2-Sample Symmetric Sampling, starting standard deviation for exploration as the standard deviation assumed for the PUFs and step sizes 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 standard normal distributed.

4.1 Standard Arbiter PUF

Model. If we take the linear delay model from section 2 into account with respect to eq. 1, the output t of this basic type of suggested electrical PUFs, the Standard Arbiter PUF (Arb-PUF), is determined by the sign of the propagation delay difference Δ.

Results. In all cases we have been able to learn the PUFs with prediction rates above 99% in 20,000 evaluations. Table 1 shows that the need of evaluations seems to grow only linearly with the number of bits. When comparing PGPE and ES, we observe that PGPE after the same number of evaluations achieves a remaining prediction error that is smaller by a factor of 5. Further, PGPE performs computationally about 4 times faster on this type of Arbiter PUF.

ES on Arb-PUFs				
	90%		95%	
Bit	E	E/Bit	E	E/Bit
16	446	27.88	720	45.00
32	878	27.44	1530	47.81
64	1879	29.36	3589	56.08
128	4230	33.05	9480	74.06

PGPE on Arb-PUFs				
	90%		95%	
Bit	E	E/Bit	E	E/Bit
16	118	7.38	190	11.88
32	219	6.84	384	12.00
64	467	7.30	834	13.03
128	1080	8.44	1890	14.77

Table 1. The evaluations needed to achieve an average prediction rate of 90% and 95% with ES and PGPE. "E" marks the columns with the average evaluations, while "E/Bit" marks the columns that shows the evaluations needed per number of bits.

4.2 XOR Arbiter PUF

In this experiments we used random subsets of 8,000 CRPs for the evaluation step of the individuals for all XOR Arbiter PUF (XOR-PUF) experiments.

Model. One possibility to strengthen the resilience of arbiter architectures against machine learning is to employ l individual Arb-PUFs, each with k stages. The same k-bit challenge $C = b_1 \cdots b_k$ is applied to each of these Arb-PUF, and their individual outputs t_i are XORed (i.e. added modulo 2) in order to produce a global output t_{XOR} [14] (see Fig. 1 b). We denote such an architecture as l-XOR-PUF. This builds a "needle in a haystack" search space that is very challenging for ES and PGPE.

Results. Figures 2 and 3 show the best of a total of 10 runs on each XOR-PUF that have been conducted. Table 2 shows the needed evaluations to achieve a prediction rate of 90% and the fraction 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 for ES and PGPE. We speculate on the basis of the obtained data that the growth rate is higher degree polynomial, but stress that due to noise and the small database, a definite and final conclusion

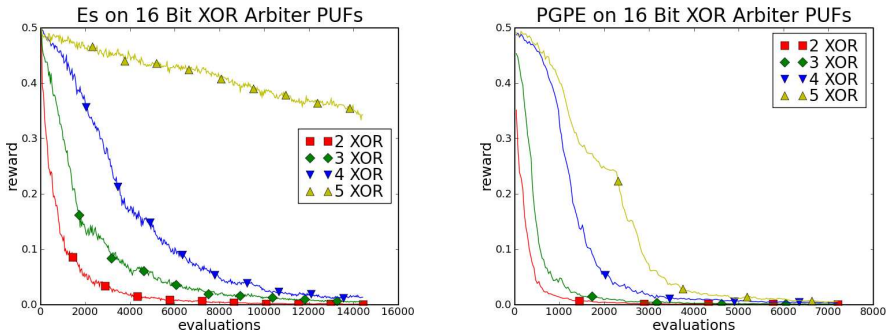

Fig. 2. The best of 10 runs on each XOR-PUF architecture with a 16 bit input vector.

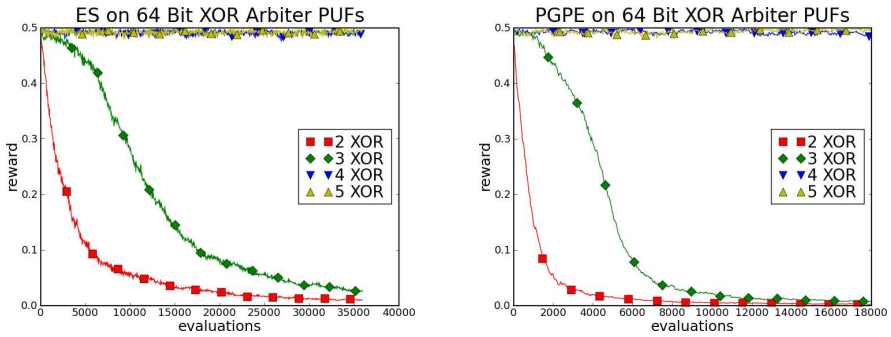

Fig. 3. The best of 10 runs on each XOR-PUF architecture with a 64 bit input vector.

is difficult. The fraction of runs that succeed in predicting the PUF with a suitable rate drops strongly 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% with both ML methods. Our experiments show that the XOR-PUF can be broken up to 3 XORs and challenge length of 64 bit. When comparing PGPE and ES, we observe that PGPE performs computationally about 3 times faster on this type of PUF. Also the success rate drops less for PGPE, so PGPE seems slightly more relieable on breaking this kind of PUF as can be seen nicely in Figure 2.

4.3 Feed Forward Arbiter PUF

The Feed Forward Arbiter PUF (FF-PUF) is 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 [7], we showed that FF-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 and therefore opens up the RL domain for attacking this type of PUF. The exact architecture of the chosen FF-PUF structures (length of loops, start and

ES on XOR-PUFs				
	E		Rate	
XOR	16 Bit	64 Bit	16 Bit	64 Bit
2	1080	5508	100%	100%
3	3031	17460	90%	50%
4	5796	-	30%	0%
5	-	-	0%	0%

PGPE on XOR-PUFs				
	E		Rate	
XOR	16 Bit	64 Bit	16 Bit	64 Bit
2	315	1350	100%	100%
3	556	5620	50%	90%
4	1690	-	40%	0%
5	2810	-	40%	0%

Table 2. The number of evaluations needed to achieve a prediction rate of 90% and the rate of runs that achieved this prediction rate in the given maximal number of evaluations with ES and PGPE. "E" marks the columns with the evaluations needed, while "Rate" marks the column that shows the rate of successful runs.

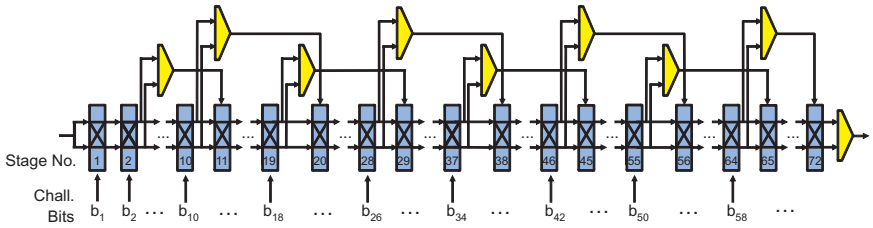

Fig. 4. The architecture of the FF-PUFs that we employed in our ML experiments, shown for the 8 FF-loop case.

end point of loops) are shown in Figure 4. This architecture of symmetric and equally distributed loops seems the most natural way of placing the loops, and has also been used in earlier publications [15],[7]; investigations are on the way to consolidate that this is indeed the optimal, i.e. hardest to learn, architecture.

Model. FF-PUFs were introduced in [9, 15, 5] and further discussed in [11]. Their basic layout is similar to the architecture of Arb-PUFs. However, some of their multiplexers are not switched in dependence of an external challenge bit, but as a function of the delay differences accumulated in earlier parts of the circuit. Additional arbiter components evaluate these delay differences, and their output bit is fed into said multiplexers in a "feed-forward loop" (FF-loop). The number of loops as well as the starting and end point of the FF-loops are variable design parameters. Please note that a FF Arb-PUF with k-bit challenges $C = b_1 \cdots b_k$ and l loops has $s = k + l$ multiplexers or stages. The described dependency makes natural architecture models of FF Arb-PUFs not differentiable any more. This architecture generates an interesting multi-modal search space that is also challenging for ES and PGPE.

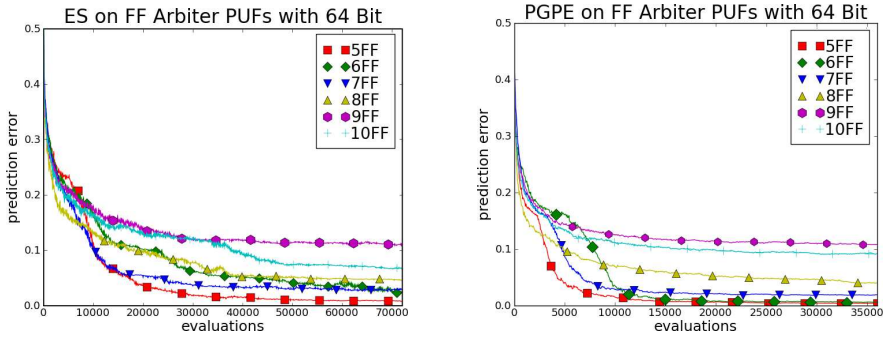

Fig. 5. The best of 40 runs (80 for 9+10 FF) with ES and the best of 10 runs (20 for 9+10 FF) with PGPE on each FF-PUF architecture.

	ES on FF-PUFs					PGPE on FF-PUFs				
	90%		95%		Best	90%		95%		Best
FF	E	E/FF	E	E/FF	Result	E	E/FF	E	E/FF	Result
5	10200	2040	17100	3420	99.3%	2900	580	3730	746	99.6%
6	21200	3533	42300	7050	97.7%	7840	1307	9340	1557	99.5%
7	10100	1443	22700	3243	97.4%	4710	673	6580	940	98.2%
8	17600	2200	53100	6638	95.5%	4840	605	21350	2669	96.1%
9	-	-	-	-	89.2%	-	-	-	-	89.3%
10	37500	3750	-	-	93.4%	15480	1548	-	-	90.9%

Table 3. The evaluations needed to achieve an prediction rate of 90% and 95% for the best run out of 40 (80 for 9+10 FF) for ES and out of 10 (20 for 9+10 FF) for PGPE. E stands for the needed number of evaluations, and E/FF symbolizes the needed evaluations divided by the number of FF loops. "Best" marks the columns with the best results after 72,000 evaluations (ES) and 36,000 evaluations (PGFPE).

Results. Figure 5 shows the best of a total of 40 runs for ES and 10 runs for PGPE on each FF-PUF that have been conducted while varying the number of FF-loops. Table 3 shows the evaluations needed to achieve an average prediction rate of 90% and 95%. As shown, in all cases we have been able to successfully learn the PUFs with rates close to 90%. 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 $45°C$, which is only 90.16% [16]. The experiments show that the FF-Arbiter are definitely susceptible to attacks by ES and PGPE. When comparing PGPE and ES, we observe that PGPE performs computationally about 3 times faster on this type of Arbiter PUF.

5 Summary and Conclusion

We investigated the performance of the recently published Policy Gradient method PGPE in the cryptanalysis of electrical PUFs. We found that up to a medium level of size and complexity, essentially all currently known electrical PUFs are susceptible to attacks by this method. One particular advantage of PGPE in the cryptanalysis of circuit-based PUFs is, that it merely requires a parametric internal model of the PUF. In opposition to that, other ML methods that have been applied for PUF attacks require linearly separable or differentiable models, which can be hard to find, or may not even exist at all. Our results further reveal that PGPE significantly outperforms Evolution Strategies in the cryptanalysis of PUFs. ES was up to now the only other known ML method applicable to all existing electrical PUFs, since it also worked on the mere basis of a parametrizable model. Due to their broad applicability, and since they do not require hard-to-optimize differentiable or separable models, PGPE-based tests have the potential of becoming a standard security benchmark for circuit-based PUFs: ML curves obtained by PGPE on small instances can help us to judge and compare the security of various electrical PUF implementations.

References

1. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science **297**(5589) (2002) 2026
2. Gassend, B., Clarke, D., van Dijk, M., Devadas, S.: Silicon physical random functions. In: ACM Conference on Computer and Communications Security-CCS. (2002) 148–160
3. Pappu, R.: Physical One-Way Functions. Phd thesis, MIT
4. Rührmair, U., Busch, H., Katzenbeisser, S.: Strong pufs: Models, constructions and security proofs, Towards Hardware Intrinsic Security: Foundation and Practice, Springer (2010)
5. Lim, D.: Extracting Secret Keys from Integrated Circuits. Msc thesis, MIT (2004)
6. Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J.: Parameter-exploring policy gradients. Neural Networks **23**(4) (2010) 551–559
7. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Stoyanova, V., Schmidhuber, J.: Machine learning attacks on physical unclonable functions. In: ACM Conference on Computer and Communications Security-CCS (to be published). (2010)
8. Schwefel, H.: Evolution and Optimum Seeking: The Sixth Generation. John Wiley & Sons, Inc. New York, NY, USA (1993)
9. Gassend, B., Lim, D., Clarke, D., Van Dijk, M., Devadas, S.: Identification and authentication of integrated circuits. Concurrency and Computation: Practice & Experience **16**(11) (2004) 1077–1098
10. Majzoobi, M., Koushanfar, F., Potkonjak, M.: Lightweight secure PUFs. In: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, IEEE Press (2008) 670–673
11. Majzoobi, M., Koushanfar, F., Potkonjak, M.: Testing techniques for hardware security. In: Proceedings of the International Test Conference (ITC). (2008) 1–10
12. Bäck, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press, USA (1996)
13. Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J.: Policy gradients with parameter-based exploration for control. In J. Koutnik V. Kurkova, R.N., ed.: Proceedings of the International Conference on Artificial Neural Networks. Volume I, LNCS 5163. (2008) 387–396
14. Suh, G., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th annual Design Automation Conference, ACM (2007) 14
15. Lee, J., Lim, D., Gassend, B., Suh, G., Van Dijk, M., Devadas, S.: A technique to build a secret key in integrated circuits for identification and authentication applications. In: Proceedings of the IEEE VLSI Circuits Symposium. (2004) 176ff
16. Lim, D., Lee, J., Gassend, B., Suh, G., Van Dijk, M., Devadas, S.: Extracting secret keys from integrated circuits. IEEE Transactions on Very Large Scale Integration Systems **13**(10) (2005) 1200