On the Scaling of Machine Learning Attacks on PUFs with Application to Noise Bifurcation

Johannes Tobisch and Georg T. Becker

Horst Görtz Institute for IT Security
Ruhr-University Bochum, Germany

Abstract. Physical Unclonable Functions (PUFs) are seen as a promising alternative to traditional cryptographic algorithms for secure and lightweight device authentication. However, most strong PUF proposals can be attacked using machine learning algorithms in which a precise software model of the PUF is determined. One of the most popular strong PUFs is the XOR Arbiter PUF. In this paper, we examine the machine learning resistance of the XOR Arbiter PUF by replicating the attack by Rührmaier *et al.* from CCS 2010. Using a more efficient implementation we are able to confirm the predicted exponential increase in needed number of responses for increasing XORs. However, our results show that the machine learning performance does not only depend on the PUF design and and the number of used response bits, but also on the specific PUF instance under attack. This is an important observation for machine learning attacks on PUFs in general. This instance-dependent behavior makes it difficult to determine precise lower bounds of the required number of challenge and response pairs (CRPs) and hence such numbers should always be treated with caution.
Furthermore, we examine a machine learning countermeasure called noise bifurcation that was recently introduced at HOST 2014. In noise bifurcation, the machine learning resistance of XOR Arbiter PUFs is increased at the cost of using more responses during the authentication process. However, we show that noise bifurcation has a much smaller impact on the machine learning resistance than the results from HOST 2014 suggest.

Key words: Physical Unclonable Function, Machine Learning Attacks, Arbiter PUF, Noise-Bifurcation

1 Introduction

Physical Unclonable Functions have gained a lot of research interest in the last years. In Physical Unclonable Functions (PUFs), the inherent process variations of a computer chip are used to give every chip a unique and unclonable identity. The first electrical PUF, the Arbiter PUF, was introduced by Gassend *et al.* in 2002 [1]. A challenge is sent to the Arbiter PUF and each PUF instance answers with a unique response. This way the Arbiter PUF can be used in a simple authentication protocol. The secret information in a PUF are the process variations of the chips and not a digital key. In theory it should not be possible to create an exact clone of a PUF and hence it should be “unclonable”. However, for existing electrical strong PUFs such as the Arbiter PUF it is possible to model the PUF in software. The parameters needed for such a software model can be approximated using machine learning techniques given enough challenge and response pairs (CRPs). Different variants of the Arbiter PUF have been proposed to increase the resistance against machine learning attacks, e.g., the Feed-Forward Arbiter PUF [2], the controlled PUF [3], the XOR Arbiter PUF [4] and the Lightweight PUF [5]. From these solutions the XOR Arbiter PUF has gained the most attention. In an XOR Arbiter PUF, the responses of several Arbiter PUFs are XORed to increase the machine learning complexity. However, while there are many papers discussing PUFs, relatively few directly examine machine learning algorithms against PUFs. The most prominent work on machine learning attacks on PUFs is the 2010 CCS paper by Rührmaier *et al.* [6]. They demonstrated that XOR Arbiter PUFs can be attacked using a Logistic Regression based machine learning algorithm. The initial results were based on simulated data, but follow-up work using silicon data confirmed the simulation results [7].

While their results showed that XOR Arbiter PUFs can be attacked using Logistic Regression, they also showed that the required number of responses grows exponentially with the number of XORs. Hence, in theory it would be possible to build an XOR Arbiter PUF that is resistant against logistic regression machine learning attacks if the PUF parameters are large enough. However, large PUF parameters contradict the lightweight nature of PUFs and each additional XOR increases the unreliability of the PUF. Hence, the number of XORs that can be used is limited in practice. In addition to different PUF constructs, several PUF protocols based on Arbiter PUFs have been proposed, such as the reverse fuzzy extractor protocol [8] or the Slender PUF protocol [9]. A good analysis of different PUF protocols and their weaknesses can be found in [10].

Recently, at HOST 2014, a new machine learning countermeasure called noise bifurcation was introduced by Yu *et al.* [11]. In noise bifurcation it is assumed that during the set-up phase a verifier builds a precise software model of the PUF that can be used to authenticate the PUF. At the cost of increasing the number of needed response bits, the machine learning resistance is increased without adding additional XORs, unreliability or area overhead, which makes this technique very interesting.

1.1 Our Contribution

In this paper we use an efficient implementation of the Logistic Regression (LR) machine learning algorithm to replicate the results presented by Rührmair et al. for PUFs with a larger number of XORs. Our results suggest that the exponential increase in CRPs needed for a LR machine learning attack claimed by Rührmair *et al.* holds for larger PUF instances. However, our results show that the success probability of a machine learning attack does not only depend on the PUF parameters and the number of responses. Instead, our results suggest that some PUF instances are easier to attack using machine learning than others. This makes it very difficult to make precise statements about the machine learning resistance of a PUF, since some PUF instances might be less resistant against machine learning attacks than others. This is an important observation that should be kept in mind when evaluating results of machine learning attacks.

Furthermore, we used our efficient implementation to examine the machine learning complexity of the noise bifurcation technique. Our results are in contrast to the results presented in [11]. While the machine learning complexity increases when noise bifurcation is used, we show that the increase due to noise bifurcation is significantly less than the results in [11] suggest.

2 Background

The main idea behind Arbiter PUFs is that the performance of every CMOS gate is slightly different, even if the exact same layout and mask are used. Hence, every CMOS gate will have slightly different timing characteristics. In the Arbiter PUF, these timing differences are used to generate a device-specific response for a given challenge. The idea is to apply a race signal to two identical signal paths. Which of the two paths is faster is determined only by the process variations, which are unique for every device. Both paths end in an arbiter, which generates the response by determining which of the two signals arrived first. In order to make the paths depend on a challenge, both signals are sent through a set of delay stages. Each of these delay stages can be configured to take two different paths. If the challenge bit for a delay stage equals zero, both signals are passed to their corresponding outputs. If the challenge bit of a delay stage is set to one, both signals are crossed. Figure 1 shows an example Arbiter PUF consisting of four stages. Each of the delay stages is realized in practice using two multiplexers as depicted in Figure 2.

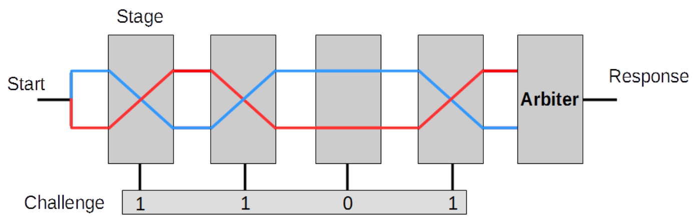


Fig. 1. Schematic of a four-stage Arbiter PUF.

2.1 Modeling an Arbiter PUF

If the internal parameters of the PUF are known it is possible to build a precise software model of an Arbiter PUF. Each delay stage adds a delay to the two

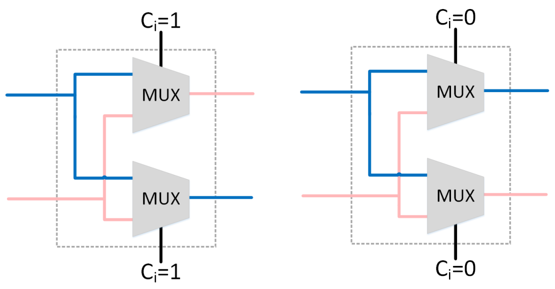


Fig. 2. Schematic of a single stage of an Arbiter PUF for both possible cases, $c_i = 1$ and $c_i = 0$.

race signals. Since we are only interested in the relative delays between the top and bottom signal, we do not need the delay added to each signal but only the added delay difference between the top and bottom signal. A delay stage i can therefore be expressed using two parameters, the delay differences $\delta_{0,i}$ and $\delta_{1,i}$, corresponding to the added delay difference when the challenge bit is '0' and '1', respectively. If these two parameters are known for every delay stage, the final delay difference for every challenge can be computed. However, the fact that the paths are crossed in stages where the challenge bit is '1' needs to be considered in this computation. Switching the top and bottom signal effectively turns a positive delay difference (the top signal is faster up to this point) into a negative delay difference (the top signal becomes the bottom signal and hence the bottom signal is now faster). Hence, switching the top and bottom signal is equivalent to changing the sign of the delay difference. The delay difference ΔD_i after the ith stage can be computed recursively according to the following equation, where c_i denotes the ith challenge bit:

$$\Delta D_i = \Delta D_{i-1} \cdot (-1)^{c_i} + \delta_{c_i,i} \tag{1}$$

The final response r can be computed given the delay difference ΔD_n after the last delay stage n by evaluating the sign of ΔD_n:

$$r = \begin{cases} 1 & \text{if} \quad \Delta D_n > 0 \\ 0 & \text{if} \quad \Delta D_n < 0 \end{cases} \tag{2}$$

However, in practice there is a more efficient method to model an Arbiter PUF with only $n+1$ parameters as opposed to the $2 \cdot n$ parameters needed in this recursive method. For this, one needs to compute the delay vector $\boldsymbol{w} = (w_1, ..., w_{n+1})$:

$$\begin{aligned} w_1 &= \delta_{0,1} - \delta_{1,1} \\ w_i &= \delta_{0,i-1} + \delta_{1,i-1} + \delta_{0,i} - \delta_{1,i} \\ w_{n+1} &= \delta_{0,n} + \delta_{1,n} \end{aligned} \tag{3}$$

Additionally, the challenge vector $\boldsymbol{c} = c_1, ..., c_n$ has to be transformed into the feature vector $\boldsymbol{\Phi} = (\Phi_1, .., \Phi_{n+1}) \in \{-1, 1\}^{n+1}$:

$$\begin{gathered} \Phi_i = \prod_{l=i}^{n} (-1)^{c_l} \quad \text{for} \quad 1 \leq i \leq n \\ \Phi_{n+1} = 1 \end{gathered} \tag{4}$$

These transformations allow us to compute the delay difference ΔD_n with a simple scalar multiplication:

$$\Delta D_n = \boldsymbol{w}^\intercal \boldsymbol{\Phi} \tag{5}$$

This representation has two tremendous advantages. First, the number of model parameters is reduced to $n + 1$ and, second, the scalar multiplication is an operation that can be implemented efficiently.

2.2 XOR Arbiter PUF

As shown by Rührmair *et al.* in [6], Arbiter PUFs are susceptible to machine learning attacks. These attacks usually require a certain amount of recorded CRPs and allow the adversary to predict the responses for new challenges. To improve the resistance against these attacks, Suh and Devadas [4] proposed a design in which the results of several Arbiter PUFs are combined by XORing them. While this additional non-linearity does not completely prevent machine learning attacks, it does increase the complexity significantly. The model for an XOR Arbiter PUF builds upon the model for the single Arbiter PUF described in the previous section. Assuming the XOR Arbiter PUF has l different Arbiter PUFs which share the same challenges $\boldsymbol{\Phi}$ and each PUF has a unique delay vector $\boldsymbol{w}_j$, then the response r_{XOR} can be computed according to the following equation:

$$\Delta D_{\text{XOR}} = (-1)^{l+1} \prod_{j=1}^{l} \boldsymbol{w}_j^\intercal \boldsymbol{\Phi} \tag{6}$$

$$r_{\text{XOR}} = \begin{cases} 1 & \text{if} \quad \Delta D_{\text{XOR}} > 0 \\ 0 & \text{if} \quad \Delta D_{\text{XOR}} < 0 \end{cases} \tag{7}$$

Please note that the absolute value of ΔD_{XOR} has no meaning in the context of the model. Only its sign carries significant information.

It should also be noted that reliability is a valid concern for XOR Arbiter PUFs. Parameters such as the supply voltage or temperature can have negative effects on the reliability of Arbiter PUFs [12]. An XOR Arbiter PUF aggregates the unreliability of its underlying Arbiter PUFs. Thus, its reliability decreases exponentially with the number of XORs.

3 Scaling of Machine Learning Attacks on XOR Arbiter PUFs

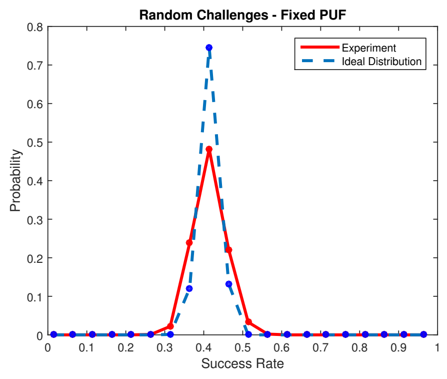
In our machine learning attacks we used logistic regression, together with the RPROP (LR-RPROP) optimization algorithm as proposed by Rührmair *et al.* [6]. In [6] the largest attacked PUF design was a 5 XOR, 128 stage Arbiter PUF using $500,000$ CRPs. In order to test how the machine learning attack scales for larger instances we made a speed and memory optimized implementation of the LR-RPROP algorithm. We used Matlab for the non-time-critical parts of the machine learning algorithm and MEX C functions for the time-critical operations such as computing the gradient of the cost function. Our experiments were conducted using an AMD Opteron cluster, which consists of 4 nodes, each with 64 cores and 256 GB of memory. To make use of the many cores available on the cluster we used OpenMP in the MEX C functions for parallel processing. Due to the diminishing return in our parallelization, we only used 16 cores for each run and instead started several attacks at the same time. Hence, up to 16 machine learning attacks were executed on the cluster[1].

We followed the approach in [6] and used one set of challenges and responses during the training phase of the machine learning algorithm and a second reference set of challenges and responses to determine the resulting model accuracy. For modeling the PUFs we assumed a Gaussian distribution of the PUF delay parameters δ and generated the PUF instance using the Matlab normrand() function. The challenges were also generated randomly. The required training set size grows with increased security parameters of the PUF, that is the number of stages and XORs. This, in turn, increases the runtime of the attack. The runtime of the whole attack is also influenced by the number of runs required. An unsuccessful run occurs when the RPROP algorithm gets stuck in a local minimum of the cost function. Such a run has *not converged*. In this case the RPROP algorithm needs to be restarted with a different start parameter. Each restart is denoted as a *run*. In our implementation we use randomly generated start parameters when restarts are required. Please note that in such a restart, the same challenges and responses are used.

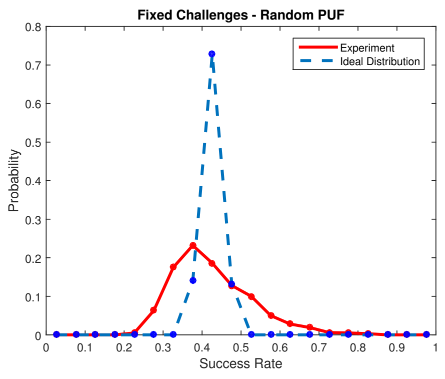
It is well known that the probability of a run not converging decreases with larger training sets [6]. However, the question is what else effects the convergence rate. To test this, in a first experiment 1000 machine learning attacks on the same PUF instance of a 4 XOR, 64 stage Arbiter PUF with $14,000$ responses were performed. A different set of challenges and responses was used in each iteration of the attack and the percentage of runs that converged, i.e., that achieved a model accuracy of more than 98%, was determined. The results can be found in Figure 3(a). The average convergence rate for this PUF instance and $14,000$ responses was 0.4, i.e., in average 200 out of the 500 runs converged. If a particular set of challenges does not have an impact on the convergence rate, then the chance that of a run converging should be the same for all runs, i.e.,

[1] Please note that we did not have exclusive access to the cluster and hence other computations were performed in parallel to our attacks.

the runs should be independent and identically distributed (i.i.d.). We call this distribution the *ideal distribution.* As a reference, the ideal distribution for 1000 attacks with 500 runs each is depicted in Figure 3(a). There is no significant difference between the ideal distribution and the results of the machine learning attack. Hence, one can assume that, if chosen randomly, the challenge set has negligible influence on the convergence rate. In the next experiment, the same challenge set was used for all 1000 attacks but a different, randomly generated PUF instance was used in each attack. The results of this experiment can be found in Figure 3(b).



(a)



(b)

Fig. 3. Histogram of 1000 machine learning attacks on a 4 XOR, 64 stage Arbiter PUF with 500 runs each. The X-axis depicts the achieved convergence rate and the Y-axis is the probability with which such a convergence rate occurred. In (a) the same PUF instance was attacked in all 1000 attacks but a different challenge set was used each time. In (b) the same challenge set but a different PUF instance was used for each attack. The ideal distribution represents the expected distribution if each attack is independent and identically distributed (i.i.d.).

One can clearly see that the convergence rate distribution of this experiment does not follow the ideal distribution anymore. Some PUF instances have a considerably lower or higher convergence rate than any trial in the ideal distribution. Hence, one can conclude that the PUF instance has direct impact on the machine learning complexity. This is a very important result. The worst and best case convergence rates can differ significantly for different instances of the same PUF design. This observation makes it difficult to determine a precise number of needed responses for a given PUF design, since this number is not consistent across different instances of the same PUF design.

To determine the machine learning complexity for different PUF designs, we aimed to limit the influence of the different instances on the results. Therefore we chose a simulation approach which uses a new PUF instance for each run. I.e., for each try we perform 100 machine learning attacks with different challenges and PUF instance in each run. This way we limit the impact of outliers, i.e., PUF instances that are extremely hard or easy to model, on our measured success rate and estimated training time. We used a test set of $10,000$ CRPs to validate the success of the learning. We counted a prediction rate of above 98% as a success. Unsuccessful tries, which did not achieve a prediction rate of above 98% on the training set, were stopped after 800 RPROP iterations. The number of RPROP iterations had to be lowered to 400 for large instances due to limited computational resources. In Table 1, the results for attacks on XOR Arbiter PUFs with different parameters are summarized. For each tested PUF construct the optimal number of challenges and responses in terms of machine learning attack time is determined. For a smaller number of CRPs, less runs converge and hence more need to be restarted. This increases the machine learning attack time. On the other hand, if more challenges and responses are used, then a single run takes longer since more challenges need to be evaluated in each step. However, if the attacker is willing to invest more computation time, he can attack the PUF instance with much fewer challenges than the optimal number. On the right side of Table 1 one can see the minimum number of challenges for which at least one of the 200 runs converged. Please note that this number is not the smallest number for which a machine learning attack can be successful. For one, an attacker might be willing to try much more than 200 runs. Furthermore, as discussed earlier, some PUF instances are easier to model than others. Hence, it might be possible to attack some PUF instances with less challenges and responses. Nevertheless, the results are a good indication of how many challenges are enough to attack the XOR Arbiter PUF with Logistic Regression. Please also note that for large PUF instances we were not able to run as many tries as for small instances and hence the numbers become less reliable for large instances. As can be seen, we were able to attack instances of up to 9 XOR, 64 stage PUFs and 7 XOR, 128 stage PUFs, whereas previously only attacks on instances of 6 XOR, 64 stage PUFs and 5 XOR, 128 stage PUFs have been published. In order to predict the machine learning complexity for increasing XORs, we tried to empirically find an approximate factor for the increase in required CRPs. As mentioned, it is hard to define the required number of challenges to attack an XOR Arbiter PUF. To get a fair comparison, we tried to determine the number of challenges for different Arbiter PUF designs so that a convergence rate of ca. 25% was reached. The results of this experiment can be seen in Table 2. Unfortunately, for large PUF instances we were not able to determine the number of challenges needed for a 25% convergence rate very accurately due to the large computational complexity of these experiments. However, the results presented in Table 2 indicate that the relative increase in number of challenges seems to be similar for each XOR. Hence, the results suggest that the predictions performed in [6] were accurate and that the needed number of challenges and responses indeed increases exponentially with the number of XORs for a LR-RPROP machine learning attack.

Based on this and the results for large instances from Table 1, we observe that it is not possible with our implementation and hardware to break either 64 stages and 10 XORs or 128 stages and 8 XORs. Assuming a multiplicative

Table 1. The needed number of challenge and response pairs (CRPs) for different XOR Arbiter PUF designs. In the left column the optimal number of CRPs that minimizes the training time is listed while in the right column the minimum number of CRPs in which at least one of the 200 runs converged is shown. The memory consumption includes the challenge and response set and the largest temporary arrays. The training time is estimated on the basis of the average time per RPROP iteration and the convergence rate. For most designs 200 runs were performed. Instances with a lower number of runs are marked with an asterisk.

Stages	XORs	optimal CRP	training time	memory	minimal CRP	training-time	memory
64	4	$42 \cdot 10^3$	2 sec	2.1 MB	$10 \cdot 10^3$	16 sec	501 KB
	5	$260 \cdot 10^3$	8 sec	15.1 MB	$45 \cdot 10^3$	2:46 min	2.6 MB
	6	$1.40 \cdot 10^6$	1:01 min	92.6 MB	$210 \cdot 10^3$	30:24 min	13.9 MB
	7^*	$20 \cdot 10^6$	54:32 min	1.5 GB	$3 \cdot 10^6$	2:43 hrs	220 MB
	8^*	$150 \cdot 10^6$	6:31 hrs	12.3 GB	$40 \cdot 10^6$	6:31 hrs	6.2 GB
	9^*	$350 \cdot 10^6$	37:46 hrs	31.5 GB	$350 \cdot 10^6$	37:46 hrs	31.5 GB
128	4	$200 \cdot 10^3$	5 sec	13.2 MB	$22 \cdot 10^3$	2:24 min	1.5 MB
	5	$2.2 \cdot 10^6$	54 sec	163.1 MB	$325 \cdot 10^3$	12:11 min	24.1 MB
	6^*	$15 \cdot 10^6$	4:45 hrs	1.2 GB	$15 \cdot 10^6$	4:45 hrs	1.2 GB
	7^*	$400 \cdot 10^6$	66:53 hrs	36.1 GB	$400 \cdot 10^6$	66:53 hrs	36.1 GB

factor of 8 for the training set size, 64 stages and 10 XORs would require roughly $2.8 \cdot 10^9$ CRPs, which results in a memory consumption of 275 GB. Similarly, 128 stages and 8 XORs would require roughly $5.4 \cdot 10^9$ CRPs or 530 GB of memory. This is more than the 256 GB of memory that our cluster provides per node. This limitation can be circumvented by adding support for distributed memory to the implementation. Hence, a dedicated attacker can very likely also attack such a PUF.

4 Machine Learning Attacks on Noise Bifurcation

In the previous section, we highlighted the susceptibility of XOR Arbiter PUFs to machine learning attacks. In this context, it is of great interest to find ways to improve the security of XOR Arbiter PUFs without adding more XORs since each additional XOR increases the manufacturing costs and unreliability of the PUF. One such possible improvement, a noise bifurcation architecture, was introduced by Yu *et al.* at HOST 2014 [11].

The main idea behind noise bifurcation is to prevent the attacker from associating challenges and their corresponding responses. Hence, the goal is to obfuscate the responses from an attacker's point of view while a verifier should still be able to authenticate the PUF. In noise bifurcation it is assumed that the verifier has a software model of the PUF under test. To achieve this, in a set-up phase the verifier collects direct responses from each of the individual Arbiter PUFs of the used XOR Arbiter PUF. Using these direct responses, the verifier can use machine learning to derive a precise software model of each individual

Table 2. Scaling of the number of challenge and response pairs (CRPs) that are required to achieve a success rate of around 0.25.

Stages	XORs	CRPs	Success Rate	CRP factor
64	4	$12 \cdot 10^3$	0.29 (58/200)	-
	5	$90 \cdot 10^3$	0.28 (56/200)	7.5
	6	$750 \cdot 10^3$	0.26 (52/200)	8.3
	7	$5 \cdot 10^6$	0.31 (10/32)	6.7
	8	$50 \cdot 10^6$	0.5 (14/28)	10
	9	$350 \cdot 10^6$	0.25 (2/8)	7
128	4	$65 \cdot 10^3$	0.26 (52/200)	-
	5	$975 \cdot 10^3$	0.26 (52/200)	15
	6	$22 \cdot 10^6$	0.25 (2/8)	22.6
	7	$400 \cdot 10^6$	0.38 (3/8)	18.2

Arbiter PUF. Once this is done, the set-up phase should be permanently disabled so that direct challenges and responses are not revealed anymore.

The authentication phase is initiated by the server who sends a master challenge C_1 to the PUF token. The PUF token generates a second random master challenge C_2 and applies C_1 and C_2 to the challenge control, which computes m PUF challenges. These m PUF challenges are then applied to the XOR Arbiter PUF to compute m response bits $y_P \in \{0,1\}^m$. However, instead of directly transmitting these m response bits to the server, the PUF device performs a *response decimation*. The m responses y_P are divided into m/d groups of d responses each. From each group one response bit is randomly selected while the others are discarded. These m/d response bits form the actual response string $y'_P \in \{0,1\}^{m/d}$ that is transmitted to the server together with C_2.

The server uses the master challenges C_1 and C_2 to compute the m corresponding challenges and uses its software model of the XOR Arbiter PUF to compute the m corresponding responses y_S. The server also sorts these responses into the m/d groups of length d as done by the PUF device. But the server does not know which response was picked for each group by the PUF device. However, if all responses within a group are identical, i.e., all bits in a group are 1 or 0, the server also knows which value the corresponding bit in y'_P should have. Hence, the server discards all groups in which the bits are not equal and only compares the groups in which all response bits are equal to the corresponding bit of y'_P.

This process is illustrated in Figure 4 for $d = 2$. If the mismatch between these bits is below a threshold, the PUF is authenticated, otherwise authentication fails. While in average 2^{d-1} of the bits in y'_p are discarded, the server can predict the remaining response bits in y'_p with 100% accuracy if the software model is 100% accurate. An attacker on the other hand has to guess which challenge was chosen by the PUF device for each group. For each response bit, the chance that the attacker guesses correctly is $1/d$. If the attacker does not guess the correct challenge, the response bit has a 50% chance of being correct. Thus, while the accuracy for the server is still 100%, the attacker has lost information and hence machine learning attacks should be harder. Please note however, that

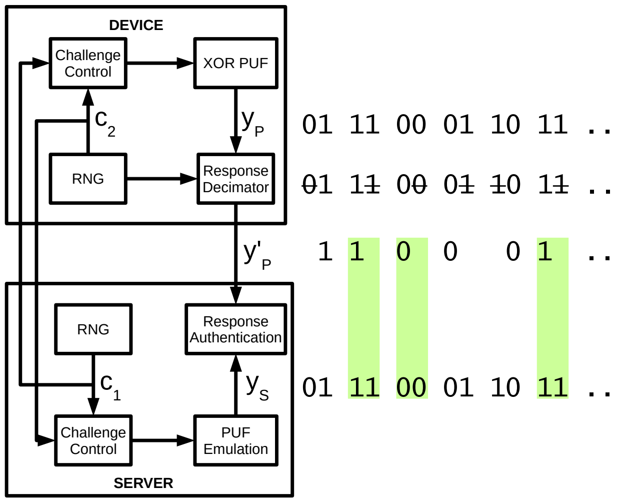


Fig. 4. Illustration of the noise bifurcation scheme with a decimation factor of $d = 2$.

the downside of noise bifurcation is that a software model is needed on the server's side. This is a major drawback of the noise bifurcation countermeasure. Furthermore, the noise bifurcation scheme would not be considered a strong PUF according to the formal definition in [13] since the challenges are generated by the device as well as the server and hence the server cannot query the PUF with the same challenge twice.

4.1 Attacking Noise Bifurcation

In order to use logistic regression, one has to build a training set from the set of m challenges and the decimated responses y'_P. Yu *et al.* mention two strategies, namely single-challenge guess and full-response replication [11]. In the first approach, the attacker guesses which challenge was used for each response bit in y'_p. This results in a training set of m/p challenges. In full-response replication, all m challenges are kept and each response bit in y'_p is duplicated $d-1$ times and associated with all challenges of the same group. We chose the full-response replication strategy for our machine learning attack, since more challenges are kept per protocol execution.

In the following we only consider the case of $d = 2$, where the m responses are divided into $m/2$ groups of size 2. We chose $d = 2$ since this is the only case for which empirical results were presented in [11]. With full-response duplication and $d = 2$, even an attacker with a 100% accurate PUF model would have a mismatch of 75% in average, since in each group of two the attacker would correctly predict one PUF response while the other PUF response would be correct with a probability of 50%. Hence, noise bifurcation with $d = 2$ should be similar to attacking an XOR Arbiter PUF which has a reliability of 75%.

4.2 Results

Yu *et al.* also used logistic regression and the RPROP algorithm to evaluate the resistance of their scheme against machine learning attacks, although some slight alteration might have been made. They used the methodology of assuming a Gaussian distribution of delay values as done in Section 3 and compared their results to those presented by Rührmair *et al.* [6]. With their implementation of the attack, which uses full-response replication, they were only able to attack small instances, i.e., 64 stages and 4 XORs. We had the chance to discuss our contradicting results with the authors of [11] and learned that for their noise bifucation implementation they used an XOR PUF whose individual Arbiter PUFs are each supplied with an independent, random challenge. This design in itself constitutes a countermeasure (cf. attacks on Lightweight PUFs [6]). Hence, their noise bifurcation implementation actually consists of two countermeasures compared to their reference XOR implementation. However, in order to understand the impact of noise bifurcation, these countermeasures need to be examined separately. In the following, we will call an XOR PUF in which the same challenge is applied to all Arbiter PUFs a *classic XOR PUF* and a PUF in which a different challenge is applied to each Arbiter PUF an *enhanced XOR PUF*.

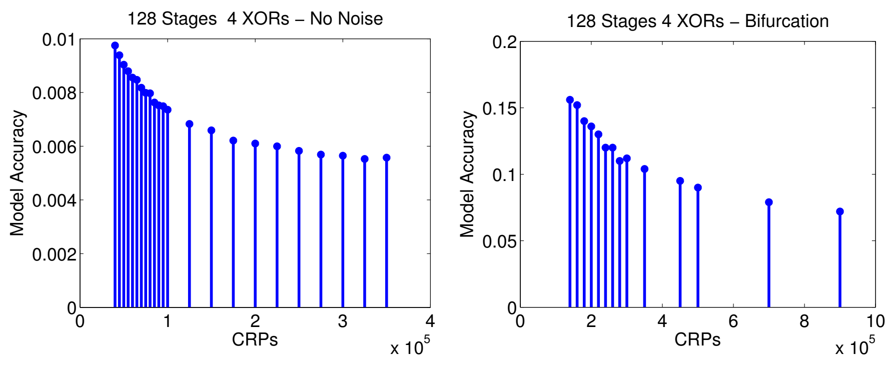
We used our implementation of the LR-RPROP machine learning algorithm from Section 3 without any additional modifications to attack the noise bifurcation countermeasure using the full-response replication strategy. As can be seen in Table 3, we were still able to break classical XOR Arbiter PUFs with noise bifurcation for up to 6 XORs with less than two million CRPs. The number of CRPs does not grow fast enough to gain a reasonable machine-learning resistance. The bigger security gain actually came from using the enhanced XOR PUF. When noise bifurcation and enhanced XOR PUF are combined, the machine learning attack complexity increased significantly. However, we were still able to break such a design with 64 stages and 5 XORs with a reasonable number of CRPs. Attacking 6 XORs does also seem possible with an estimated training set size of $300 \cdot 10^6$ CRPs.

To determine the impact of noise bifurcation on machine learning resistance, we also attacked different PUF instances with a range of varying training set sizes, as can be seen in Table 4. A lower number of CRPs increases the training time because a lot more runs are necessary, as has been discussed in Section 3. This also negatively affects the prediction error on the test set, which can be seen in greater detail in Figure 5(b). The positive trade-off between an increased number of CRPs and a lowered test set error is clearly visible. This effect is also visible for regular XOR Arbiter PUFs without noise, albeit on a smaller scale. As depicted in Figure 5(a), an increase of the training set size is met with a decreased prediction error. However, in this case, the effect of the decrease can be neglected in regard to the very low error rate. In general, the achieved model accuracy in the noise bifurcation attack is significantly lower than in the attack on plain XOR Arbiter PUFs.

The results clearly show that noise bifurcation can be attacked using machine learning and that the increase in needed challenges and responses is not as great

Table 3. Comparison of machine learning attacks on noise bifurcation with results from Yu *et al.* [11] using 64-stage PUFs with respect to the needed number of CRPs. In this table CRPs denote the number of CRPs that are computed by an attacker in the full-response replication strategy as opposed to the number of transmitted CRPs, which are actually half that number. Columns with *classic* denote experiments in which a classic XOR PUF is used while columns with *enhanced* denote experiments in which an enhanced PUF is used. In columns with *noise* the noise bifurcation countermeasure with $d = 2$ was used, i.e., the equivalent of 25% noise was added. The reference values were originally published by Rührmair *et al.* [6] and replicated in [11].

XORs	Yu et al. Reference classic	Yu et al. enhanced + noise	Our Results classic + noise	Our Results enhanced + noise
4	12,000	$1 \cdot 10^6$	$25 \cdot 10^3$	$1 \cdot 10^6$
5	80,000	$>> 2 \cdot 10^6$	$200 \cdot 10^3$	$15 \cdot 10^6$
6	200,000	$>> 2 \cdot 10^6$	$1.5 \cdot 10^6$	-



(a) (b)

Fig. 5. The average prediction error for successfully attacked instances of a 4 XOR, 128 stages Arbiter PUF in relation to the size of the training set. In (a) the plain XOR Arbiter PUF is used, while in (b) the results for an XOR Arbiter PUF in conjunction with noise bifurcation with a decimation factor of $d = 2$ are shown.

if it is not combined with other countermeasures. Nevertheless, the machine learning complexity does increase when noise bifurcation is used compared to plain XOR Arbiter PUFs, although at a much smaller scale. If this increase in machine learning complexity is worth the overhead is an open question. It is clear, however, that noise bifurcation can only be used with PUFs that already have an extremely high machine learning resistance.

5 Conclusion

In this paper we presented a detailed analysis of Logistic Regression machine learning attacks on XOR Arbiter PUFs. We used an efficient implementation to attack larger XOR Arbiter PUFs than has been done previously. Our results show that the machine learning complexity indeed increases exponentially, as

Table 4. Our results of the attack on noise bifurcation for different PUF instances and training set sizes. Target is a classical XOR PUF (same challenge for all Arbiter PUFs) with noise bifurcation.

Stages	XORs	CRPs	Mean Runs	Training Time	Test Error
64	4	$25 \cdot 10^3$	29.7	8 min	23%
64	4	$110 \cdot 10^3$	1	<2 min	13%
64	5	$200 \cdot 10^3$	33.1	22 min	14%
64	5	$340 \cdot 10^3$	2.3	<5 min	12%
64	6	$1.5 \cdot 10^6$	15.1	150 min	11%
64	6	$4 \cdot 10^6$	2.4	60 min	12%
128	4	$140 \cdot 10^3$	14.2	5 min	15%
128	4	$240 \cdot 10^3$	2.5	<5 min	12%
128	5	$2 \cdot 10^6$	11.4	144 min	8%
128	5	$4 \cdot 10^6$	2	54 min	7%

predicted by Rührmair *et al.* [6]. We showed that even a 7 XOR, 128 stages PUF can be attacked in less than 3 days using 16 cores of our cluster. Hence, only very large XOR PUF instances could withstand a dedicated attacker.

A very important result in this paper is that not only the PUF design and the number of used responses impact the machine learning algorithm. Some PUF instances are considerably more or less resistant against machine learning attacks than others. Together with the probabilistic nature of these machine learning attacks, this makes it extremely difficult to determine the minimum number of CRPs needed to attack a PUF design. If the goal is to find a strict lower bound on the number of needed CRPs, e.g., to evaluate the resistance of PUF protocols which restrict the number of responses an attacker can collect from the same PUF, it is not enough to test a few PUF instances. Even if all of the tested PUF instances cannot be attacked with a certain number of CRPs, it might be possible that there are some PUF instances that can be attacked. Furthermore, there is a clear trade-off between computational complexity and the number of CRPs. That is, the smaller the number of CRPs, the smaller the convergence rate. Hence, if an attacker is willing to use more computational power and thus to restart the machine learning algorithm more often, the attacker can be successful with a smaller number of CRPs. Together with the fact that the convergence rate also depends on the PUF instance and not only on the design, this makes it extremely difficult to define a minimum number of CRPs an attacker needs to collect for a successful machine learning attack.

Last but not least we showed in this paper that the impact of the recently introduced noise bifurcation countermeasure is significantly smaller than claimed. We could attack a classical 5 XOR, 128 stage Arbiter PUF using 2 million challenges, which corresponds to 1 million transmitted responses in the protocol, in less than three hours. Considering that the number of responses transmitted per protocol execution increases by a factor of 2^{d-1}, it is questionable if this increase in machine learning resistance is worth the introduced overhead. The method does not seem to be suited to turn a PUF design that is hard to attack using machine learning into a design that is computationally infeasible to attack.

References

1. B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon Physical Random Functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 148–160, New York, NY, USA, 2002. ACM.
2. J.W. Lee, D. Lim, B. Gassend, G.E. Suh, M. van Dijk, and S. Devadas. A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications. In *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*, pages 176–179, June 2004.
3. B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Controlled Physical Random Functions. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 149–160, 2002.
4. G.E. Suh and S. Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 9–14, June 2007.
5. M. Majzoobi, F. Koushanfar, and M. Potkonjak. Lightweight Secure PUFs. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '08, pages 670–673, Piscataway, NJ, USA, 2008. IEEE Press.
6. U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling Attacks on Physical Unclonable Functions. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 237–249, New York, NY, USA, 2010. ACM.
7. U. Rührmair, J. Sölter, F. Sehnke, Xiaolin Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas. PUF Modeling Attacks on Simulated and Silicon Data. *Information Forensics and Security, IEEE Transactions on*, 8(11):1876–1891, Nov 2013.
8. A. Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A. Sadeghi, I. Verbauwhede, and C. Wachsmann. Reverse Fuzzy Extractors: Enabling Lightweight Mutual Authentication for PUF-Enabled RFIDs. In *Financial Cryptography and Data Security*, pages 374–389. Springer, 2012.
9. M. Majzoobi, M. Rostami, F. Koushanfar, D.S. Wallach, and S. Devadas. Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 33–44, May 2012.
10. J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede. Secure Lightweight Entity Authentication with Strong Pufs: Mission Impossible? In *Cryptographic Hardware and Embedded Systems–CHES 2014*, pages 451–475. Springer, 2014.
11. M.-D. Yu, I. Verbauwhede, S. Devadas, and D. M'Raíhi. A Noise Bifurcation Architecture for Linear Additive Physical Functions. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2014)*, pages 124–129, 2014.
12. B. Gassend, D. Lim, D. Clarke, M. van Dijk, and S. Devadas. Identification and Authentication of Integrated Circuits: Research Articles. *Concurr. Comput. : Pract. Exper.*, 16(11):1077–1098, September 2004.
13. F. Armknecht, R. Maes, A. Sadeghi, F.-X. Standaert, and C. Wachsmann. A Formalization of the Security Features of Physical Functions. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 397–412, May 2011.