On the Pitfalls of using Arbiter-PUFs as Building Blocks

Georg T. Becker

***Abstract*—Physical Unclonable Functions (PUFs) have emerged as a promising solution for securing resource-constrained embedded devices such as RFID-tokens. PUFs use the inherent physical differences of every chip to either securely authenticate the chip or generate cryptographic keys without the need of non-volatile memory. Securing non-volatile memory and cryptographic algorithms against hardware attacks is very costly and hence PUFs are believed to be a good alternative to traditional cryptographic algorithms and key generation on constrained embedded devices. However, PUFs have shown to be vulnerable to model building attacks if the attacker has access to challenge and response pairs. In these model building attacks, machine learning is used to determine the internal parameters of the PUF to build an accurate software model. Nevertheless, PUFs are still a promising building block and several protocols and designs have been proposed that are believed to be resistant against machine learning attacks. In this paper we take a closer look at a two such protocols, one based on reverse fuzzy extractors [10] and one based on pattern matching [15], [17]. We show that it is possible to attack these protocols using machine learning despite the fact that an attacker does not have access to direct challenge and response pairs. The introduced attacks demonstrate that even highly obfuscated responses or helper data can be used to attack PUF protocols. Hence, our work shows that even protocols in which it would be computationally infeasible to compute enough challenge and response pairs for a direct machine learning attack can be attacked using machine learning.**

***Index Terms*—Physical Unclonable Functions, Machine Learning, Reverse Fuzzy Extractor, Evolution Strategies**

I. Introduction

Physical Unclonable Functions (PUFs) have gained wide-spread attention in the research community as a new cryptographic primitive for hardware security applications. PUFs make use of the fact that two manufactured computer chips are never completely identical due to process variations. A PUF exploits these process variations to build a unique identity for every chip. There are many applications for which PUFs can be used. Two prominent examples are its use in challenge-and-response protocols to authenticate devices as well as for secure key generation and storage. The advantage of using a PUF to generate cryptographic keys is that the PUF ensures that each chip will have its own unique secret without the need to program it first. Furthermore, securely storing a cryptographic key in embedded devices in a way that they are resistant to physical attacks such as probing,reverse-engineering and side-channel attacks is extremely difficult. Using PUFs, no key needs to be stored in non-volatile memory since the secret is instead derived from internal physical characteristics which are hard to measure from the outside.

G.T. Becker is with the Horst Görtz Institute for IT-Security, Ruhr University Bochum, Germany. (E-mail:Georg.Becker@rub.de)

A PUF usually gets a challenge and answers with a response that depends on its process variation. PUFs can be classified into two categories: *weak PUFs* and *strong PUFs*. In a weak PUF, the number of challenges the PUF can accept is very limited so that an attacker can try all possible challenges and store their corresponding responses. This way an attacker could easily forge the PUF by replacing the PUF with a simple memory look-up. A strong PUF on the other hand has a challenge space that is large enough so that it is computationally infeasible to try and store all possible challenges. Strong PUFs can be used in challenge-and-response protocols as well as for secure key generation. A weak PUF cannot be used for challenge-and-response protocols, but can still be used for secure key generation. Note that the terminology strong PUF and weak PUF might falsely give the impression that a strong PUF is "better" than a weak PUF. However, this terminology only defines the challenge space without judging the PUFs reliability, uniqueness or other security properties.

Current strong PUF designs face two big problems that are related: they suffer from unreliability [12] and are prone to machine learning attacks [18], [19]. In an ideal case, a PUF always generates the same response for a given challenge. However, due to environmental effects and thermal noise, the response to the same challenge can vary. In practice, PUFs protocols therefore either need to allow for a few false response bits or need error correction codes to correct the faulty responses. The second problem is that even strong PUFs can be modeled in software and the needed parameters to model a specific PUF instance can be determined using machine learning techniques if challenge and response pairs are known to the attacker [18].

To overcome this problem, new protocols and designs have been proposed that are believed to be resistant against machine learning attacks. Furthermore, some of these protocols actually make use of the fact that model building attacks on delay based PUFs are possible so that one party can build software models of the PUF. During a set-up, phase challenge and response pairs are revealed and an accurate software model of the PUF is constructed using machine learning techniques. After the set-up phase,

direct access to the PUF is permanently disabled and an authentication protocol is used that does not directly reveal the challenge and response pairs. Two prominent examples of PUF based authentication protocols are the reverse fuzzy extractor based protocol by van Herrewege *et al.* [10] and a pattern matching based protocol by Majzoobi *et al.* [15], [17]. These PUF protocols can be implemented very efficiently in terms of area and power. Hence, they are very promising alternatives to traditional cryptography for constrained devices such as RFID tokens or medical implants.

A. Our Contribution and Outline

In this paper we take a closer look at two prominent PUF protocols that are supposedly resistant against machine learning attacks, the reverse fuzzy extractor based protocol by van Herrewege *et al.* [10] and the Slender PUF protocol based on pattern matching [15], [17]. Despite the security claims, we show that it is possible to attack both protocols using an evolution strategy (ES) based machine learning attack. It was shown using empirical tests that given a certain number of challenge and responses a PUF can be modeled with a certain accuracy. However, a common mistake that was also done in these protocols is that the false conclusion is drawn from these empirical tests: It is assumed that to attack a PUF a certain number of direct challenge and responses are needed. However, while such tests might tells us the model accuracy that can be achieved if we have a certain number of direct challenge and responses, it does not mean that we *need* direct challenge and responses for machine learning attacks. In this paper we demonstrate, by attacking the Slender PUF protocol and the reverse fuzzy extractor protocol, that other information than direct responses can be used instead. In both cases only obfuscated responses in the form of a padded substring or helper data of an error correction code were used to perform successful machine learning attacks. The attack on the reverse fuzzy extractor protocol also shows that not only information about the value of response bits can be used for attacking a protocol, but also the information about the reliability of response bits. Since this information is often provided by the helper data of error correction codes, this attack is of importance for many different protocols and systems.

In the next Section an introduction to Arbiter PUFs is given and the ES-based machine learning algorithm which is used to attack the PUF protocols is introduced. In Section III two machine learning attacks on the reverse fuzzy extractor protocol are described: One machine learning attack that directly uses eavesdropped helper data and one attack that uses the reliability information provided by the helper data when the same challenges are used more than once. Section IV shows that both the conference as well as the journal version of the Slender PUF protocol can be attacked using ES-based machine learning attacks. The implications of these attacks are summarized in the conclusion.

II. Background

The Arbiter-PUF is the most popular electrical strong PUF in the literature and most PUF protocols are based on Arbiter PUFs or similar structures.

A. Arbiter PUF

The basic idea of the Arbiter PUF [13] is to apply a race signal to two identical delay paths and determine which of the two paths is faster. The two paths have an identical layout so that the delay difference ΔD between the two signals mainly depends on process variations. This dependency on process variations ensures that each chip will have a unique delay behavior. The Arbiter PUF gets a challenge as its input which defines the exact paths the race signals take. Figure 1 shows the schematic of an Arbiter PUF. It consists of a top and bottom signal that is fed through delay stages. Each individual delay stage consists of two 2-bit multiplexers (MUXes) that have identical layouts and that both get the bottom and top signals as inputs. If the challenge bit for the current stage is '1', the multiplexers switch the top and bottom signals, otherwise the two signals are not switched. Each transistor in the multiplexers has a slightly different delay characteristic due to process variations and hence the delay difference between the top and bottom signal is different for a '1' and a '0'. This way, the race signal can take many different paths: an n-stage Arbiter PUF has 2^n different paths the race signals can take. However, challenges that only differ in a few bits have a very similar behavior so that an Arbiter PUF does not necessarily have 2^n unique challenges. An Arbiter at the end of the PUF determines which of the two signals is faster. The Arbiter consists of two cross-coupled AND gates which form a latch. The Arbiter has an output of '1' if the top signal arrives first and '0' if the bottom signal is the first to arrive. The Arbiter can have a slight bias so that the PUF result might be slightly biased towards '0' or '1'.

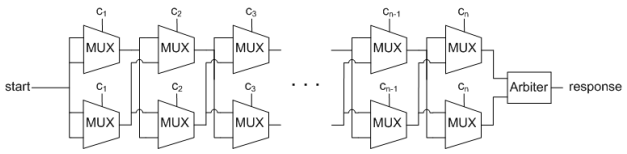

Fig. 1. Schematic of an n-bit Arbiter PUF.

To increase the resistance of Arbiter PUFs against machine learning attacks it is proposed to add a non-linear element to the PUF design. One of the most common methods to add non-linearity to a PUF design is the XOR Arbiter PUF. In an k-XOR Arbiter PUF, k PUF instances are placed on the chip. Each of the PUF instances gets the same challenge and the responses of the k PUFs are XORed to build the final response bits. While the machine learning resistance increases by XORing more PUFs, adding additional PUF instances obviously also increases the area overhead of the design. Furthermore, the XOR PUFs become increasingly unreliable the more

PUFs are XORed. Hence, in practice only a small number of PUFs can be used to build an XOR Arbiter-PUF.

B. Modeling an Arbiter PUF

The response of an n-stage Arbiter PUF is determined by the delay difference between the top and bottom signal. This delay difference is simply the sum of the delay differences of the individual stages. The delay difference of each stage depends on the corresponding challenge. Hence, there are two delay differences per stage, one corresponding to a challenge of '1' and one of '0'. This way the PUF can be modeled using $2*n$ parameters. However, there exists a more efficient, way of modeling an n-stage Arbiter PUF using only $n+1$ parameters. A PUF instance is described by the delay vector $\vec{w} = (w_1, ..., w_{n+1})$ with:

$$w_1 = \delta_{0,1} - \delta_{1,1}, \tag{1a}$$

$$w_i = \delta_{0,i-1} + \delta_{1,i-1} + \delta_{0,i} - \delta_{1,i} \text{ for } 2 \leq i \leq n, \tag{1b}$$

$$w_{n+1} = \delta_{0,n} + \delta_{1,n} \tag{1c}$$

The delay difference ΔD_n at the end of the Arbiter is the result of the scalar multiplication of the transposed delay vector $\vec{w}$ with a feature vector $\vec{\Phi}$ that is derived from the challenge c:

$$\Delta D_n = \vec{w}^T \vec{\Phi} \tag{2}$$

The feature vector $\vec{\Phi}$ is derived from the challenge vector $\vec{c}$ as follows:

$$\Phi_i = \prod_{l=i}^{n} (1 - 2c_l) \text{ for } 1 \leq i \leq n$$

$$\Phi_{n+1} = 1$$

Modeling a PUF in this way can significantly decreases the simulation time and also reduces the parameters that need to be known to $n+1$. It was shown in the past how these parameters can be computed (approximated) easily using different machine learning techniques. In practice, only a few hundred challenge and response pairs are needed to model an Arbiter PUF with a predication rate very close to the reliability of the attacked PUF [11].

C. Evolution Strategies

Evolution Strategies (ES) is a widely used machine learning techniques that gets its inspiration from the evolution theory. In evolution, a species can adapt itself to environmental changes by means of natural selection, also called *survival of the fittest*. In every generation, only the fittest specimen survive and reproduce, while the weak specimen die and hence do not reproduce. Since the specimen of the next generation inherit the genes of the fittest specimen of the previous generation, the species continuously improves.

In ES-based machine learning attacks on PUFs, the same principle of survival of the fittest is used. As discussed in the previous Section, a PUF instance can be described by its delay vector w. The goal of a machine learning attack on an Arbiter PUF is to find a delay vector w that most precisely resembles the real PUF instance. The main idea of an ES machine learning attack is to generate random PUF instances and check which of these PUF instances are the fittest, i.e. which PUF instances resemble the real PUF model the best. These fittest PUF instances are kept as *parents* for the next *generation* while the other PUF instances are discarded. In the next generation, *children* are generated using the parent's delay vectors together with some random *mutations*, i.e., some random modifications of the delay vectors. From these child instances the fittest instances are determined again and kept for the next generation as parent instances. This process is repeated for many generations in which the PUF instances gradually improve and resemble the real PUF behavior more and more.

To be able to perform an ES machine learning attack two requirements are needed: 1) It needs to be possible to described a PUF instance by a vector w and 2) a fitness test is needed that, given the delay vectors w, can determine which instances are the fittest.

Since Arbiter based PUFs can be modeled using the delay vector w, whether or not an ES machine learning attack is feasible depends on requirement 2), whether or not a fitness test for these PUF models exist. Typically, the used fitness test for an Arbiter PUF is the model accuracy between the l measured responses R from the physical PUF and the computed responses R' of the PUF instance under test:

$$A = \frac{HD(R', R)}{l} \tag{3}$$

The PUF instances with the highest model accuracies are considered the *fittest*. This fitness test can be used whenever the attacker has access to challenge and response pairs.

There exists many variants of ES-machine learning algorithms that mainly differ in how many parents are kept in each generation, how the children are derived from the parents and how the random mutation is controlled. Typically, the mutation is done my adding a random Gaussian variable $N(0, \sigma)$ to each parameter. Different approaches exist how the mutation parameter σ is controlled. The closer the PUF instances are to the optimal solution, the smaller σ should be. One approach to control σ is to deterministically decrease σ in every generation. In self-adaption on the other hand, the mutation parameters adapt themselves depending on how the machine learning algorithm is currently performing. In this paper we use the CMA-ES machine learning algorithm with the default parameters [9]. CMA-ES uses recombination, i.e., one child instance depends on several parent instances. It also uses self-adaption, i.e. the mutation strength is not controlled deterministically but adapts itself depending on how the ES algorithm is performing. We chose CMA-ES since due to the self-adaption one does not need to carefully select the parameters and it performed very well in our tests compared to other ES algorithms.

III. Attacking Reverse Fuzzy Extractor PUF protocol

Van Herrewege *et al.* proposed a PUF based mutual authentication protocol based on a so called *reverse fuzzy extractor* [10]. The protocol has many similarities to a controlled PUF [7]. The main idea is to not directly reveal the PUF responses during the authentication phase. Instead, only the helper data of an error correction code of the PUF response is transmitted to the verifier. By only revealing the helper data, the protocol is supposed to be resistant against machine learning attacks. However, we will show that it is possible to use this helper data to attack the PUF protocol. In the following, we will first introduce the reverse fuzzy extractor protocol and then discuss possible attacks on the design.

A. Reverse Fuzzy Extractor Protocol

The protocol's security is based on building a *secure sketch* using a fuzzy extractor. Since PUF responses can be unreliable, fuzzy extractors and secure sketches have been proposed for secure and reliable PUF based key generation [6]. Van Herrewege *et al.* extended this idea to build a reverse fuzzy extractor that can be used in a very lightweight challenge and response protocol. Fuzzy extractors are built on error correction codes. An error correction code typically consists of two functions, a generation function $h = Gen(r)$ that generates the helper data h for a PUF response r and a reproduction function $r = Rep(h, r')$ that recovers the response r given the helper data h and a noisy response r'. In a controlled PUF, both *Gen* as well as *Rep* are executed on the PUF-token. However, *Rep* is usually computationally expensive. The key idea of the reverse fuzzy extractor is to avoid the need of the computational expensive *Rep* function on the token side and outsource it to the verifier. This makes the protocol considerably more lightweight and a very promising solution for constrained devices such as an RFID tag.

The protocol is a mutual authentication protocol, i.e., the two participating parties authenticate each other. The two parties in this protocol are the *token* with the embedded PUF and the *verifier*. The protocol is based on a generation function $Gen()$ and reproduction function *Rep*. Given a PUF response string r, the token computes the helper data $h = Gen(r)$. This helper data can be used by the verifier with a noisy response r' to recover r using the reproduction function *Rep* with $r = Rep(h, r')$ as long as the hamming distance between r and r' is below the error correction threshold t. If the response r' is too noisy, i.e. $HD(r, r') > t$, the reproduction phase can fail and $r \neq Rep(h, r')$.

In [10], the syndrome construction of the BCH(n,k,t) error correction code with n=255, k=21, and t=55 is used for the generation phase. BCH is a very common error correction code that has been proposed for various PUF applications before, e.g. in [21], [8], [1], [14]. The syndrome construction consists of a matrix multiplication of an n-bit PUF response r with the transpose of the $n \times (n-k)$ parity check matrix H of the used BCH error correcting code.

$$h = Gen(r) = r \times H^T \tag{4}$$

The BCH(255,21,55) error correcting code can correct up to 55 erroneous bits of a $n = 255$ bit PUF response using a $n-k = 234$ bit helper data h. In the reproduction function, an error vector e is computed by decoding the syndrome $s = h - r' \times H^T$ using the decoding algorithm of the used BCH code. This error vector e is subtracted from r' to recover r.

$$s = h - r' \times H^T \tag{5a}$$
$$e = Dec(s) \tag{5b}$$
$$r = r' - e \tag{5c}$$

Due to the special form of the parity check matrix H of the BCH code, the matrix multiplication $r \times H^T$ can be computed very efficiently using a single LFSR. This makes the generation function extremely lightweight in hardware. In contrast, the decoding of the syndrome s is computationally much more complex. However, the decoding is only needed for the reproduction function and is outsourced from the computationally restricted token to verifier. This is the key feature that makes the reverse fuzzy Extractor more lightweight than e.g. a controlled PUF that uses BCH for error correction.

The protocol consists of two phases, an initialization phase that is used once to set up the protocol and an authentication phase. In the initialization phase the verifier generates random challenges c_i and sends these to the token. The token computes the responses $r_i = PUF(c_i)$ and directly sends these responses to the verifier who stores them in a database. At the end of the initialization phase, the initialization phase is permanently disabled so that the token never again directly reveals challenge and response pairs.

The authentication protocol is depicted in Table I. The authentication process is started by the verifier with an authentication request to the token. The token replies with an ID and the verifier then chooses a random nonce N and selects q challenge and response pairs c_i, r_i from the database. The verifier sends the q challenges c_i together with the nonce N to the token. The token computes the responses r'_i using its PUF, i.e. $r'_i = PUF(c_i)$. In the next step the token computes the syndromes $h_i = Gen(r'_i)$ using the generation function *Gen*. The token computes the hash $a = Hash(ID, N, r'_1, .., r'_q, h_1, .., h_q)$ and transmits a and $h_1, .., h_q$ to the verifier. The verifier computes r'_i using the helper data h_i and the stored responses r'_i with $r'_i = Rep(r_i, h_i)$. If $a \neq Hash(ID, N, r'_1, .., r'_q, h_1, .., h_q)$ the verifier rejects the authentication and aborts. Otherwise the verifier computes $b = Hash(a, r'_1, ..., r'_q)$ and sends b to the prover. The prover accepts the authentication if $b = Hash(a, r'_1, ..., r'_q)$.

TABLE I
REVERSE FUZZY EXTRACTOR PROTOCOL

Token ID, physical PUF		**Verifier** $ID', (c_1, r_1), .., (c_q, r_q))$
	$\xleftarrow{auth}$	
	$\xrightarrow{ID}$	if $ID' \neq ID$ reject and abort $N \in_{\mathbf{R}} \{0,1\}^l$
	$\xleftarrow{c_1,..,c_q,N}$	
$r'_i \leftarrow PUF(c_i)$ $h_i \leftarrow Gen(r'_i)$		
$a \leftarrow Hash(ID, N, r'_1, .., r'_q, h_1, .., h_q)$	$\xrightarrow{h_1,..,h_q,a}$	
		$r'_i \leftarrow Rep(r_i, h_i)$ $a' \leftarrow Hash(ID, N, r'_1, .., r'_q, h_1, .., h_q)$ if $a' \neq a$ reject and abort $b \leftarrow Hash(a, r'_1, .., r'_q)$
	$\xleftarrow{b}$	
if $Hash(a, r'_1, .., r'_q) \neq b$ reject and abort		

B. Discussion

The security of the protocol relies on the fact that the syndrome construction is a *secure sketch* as defined in [3]. For every syndrome h there are 2^k possible responses r with $h = Gen(r)$. Each of the responses is equally likely. Therefore, an attacker cannot recover direct challenge-and-response pairs with a probability higher than 2^k for a given h. This is also true if the attacker as access to multiple different syndromes for noisy response of the same challenge. The reason for this is that multiple syndromes do not carry information about the value of the response bits, but only the positions of the bit errors. This means that given multiple helper data for noisy responses, the attacker only learns which bits have flipped, but not the value of the flipped bits. Therefore, an attacker cannot determine the response for a challenge even if he has access to multiple different helper data for the same challenge.

Since it is impossible to recover the correct response from helper data from the same challenge, the protocol is assumed to be secure against model building attacks [10]. However, we will show that the protocol can still be attacked. The main reason is that the starting assumption, that for a model building attack an attacker needs to know challenge and response pairs turns out to be wrong. The helper data leaks enough information to attack the PUF using an ES-based machine learning attack. Furthermore, while multiple different helper data for the same challenge do not reveal the response, they indicate which response bits are unstable. Using the unreliability information to attack Arbiter PUFs has recently been proposed as a fault attack in [2]. Hence, the unreliability information provided by BCH codes can be used to model arbiter PUFs using machine learning.

The security analysis only considered how much information was leaked by helper data from a single challenge and did not consider *related* challenges. In the reverse fuzzy extractor protocol [10] an LFSR is used to generate the individual n subchallenges from the master challenge. However, this approach to generated subchallenges is very problematic, as recently pointed out by Delvaux *et al.* in [4]. An attacker can send *related challenges* to the token to be able to recover the response bits. A single challenge c_1 actually consists of 255 64-bit subchallenges $c_{1,1}, c_{1,2}, .., c_{1,255}$. These subchallenges are computed using a 64-bit LFSR with $c_{1,1}$ being equal to the initial state of the LFSR. For each subchallenge, the LFSR is clocked 64 times. Assume the attacker has sent challenge $c_{1,1}$ as a master challenge to the token. The token will then use $c_{1,1}, c_{1,2}, .., c_{1,255}$ as the subchallenges to compute $r_1 = r_{1,1}, r_{1,2}, .., r_{1,255}$ and the corresponding helper data $h_1 = r_1 \times H^T$.

In the next step the attacker sends challenge $c_{1,2}$ as the master challenge to the token. The token will now use the challenges $c_{1,2}, c_{1,3}, .., c_{1,256}$ to get response $r_{1,2}, r_{1,3}, .., r_{1,256}$ and the corresponding helper data h_2. This gives the attacker a system of linear equations with 256 unknowns and $255 \times 2 = 510$ equations:

$$h_1 = (r_{1,1}, r_{1,2}, .., r_{1,255}) \times H^T \tag{6a}$$

$$h_2 = (r_{1,2}, r_{1,3}, .., r_{1,256}) \times H^T \tag{6b}$$

The attacker can simply solve this over-defined system of liner equations to recover the response bits $r_{1,1}, .., r_{1,256}$. Hence, due to the challenge generator an attacker can very easily compute the challenge and responses by sending a second related challenge to the token. In practice, PUF responses might be unstable so that a few errors might occur. However, this makes the attack only slightly more difficult and an attacker can average over multiple responses to reduce or eliminate these errors.

We would like to note that this problem is due to the specific implementation of the challenge generator. In particular, this problem occurred because the authors did not consider chosen-challenge and related challenge attacks. LFSRs are very popular for challenge generation due to their lightweight nature. However, this attack illustrates how dangerous it can be if the challenge generator is only chosen for performance reason. A good challenge generator should make it computationally infeasible for an attacker to apply related challenges to the PUF. Since the

reverse fuzzy protocol uses a hash function, one possible fix could be to use this hash function with secure padding to generate the subchallenges.

C. Direct ES-machine learning attacks on helper data

In this Section we will discuss how to attack the protocol using an ES-based machine learning attack with the helper data as input. Recall that in an ES machine learning attack we need a fitness test that can determine which of a given set of PUF models resembles the correct PUF model the best. Typically, known challenge and response pairs are used to compute the model accuracy which then serves as a fitness metric. However, the direct responses are not available in the reverse fuzzy extractor protocol and we therefore need a different fitness test based on the helper data.

Assume that the PUF models we test in our ES machine learning algorithm have an accuracy large enough so that the hamming distance between the modeled response r' and the correct response r is $HD(r', r) < t$. Then an attacker can compute the syndrome $h' = Gen(r')$ and compute the error vector $e = Dec(h - h')$. This error vector e directly reveals the hamming distance between r and r' since $e = r - r'$. Hence, once the PUF models are accurate enough that the hamming distance between the modeled PUF responses and the measured PUF responses is below t, it is easy to determine the fitness of the PUF models.

The question is if we can also find a fitness test for PUF models with hamming distances larger than t. Our solution is rather simple. Assume that we have l challenges c_i and their corresponding helper data h_i. For every helper data h_i we compute all $2^k = 2^{21}$ responses $r_{i,j}$ for which $h_i = Gen(r_{i,j}) = r_{i,j} \times H^T$ holds true. Then we compute the modeled responses r'_i using the delay vector w of the PUF model PUF' under test with $r'_i = PUF'(c_i)$. In the next step we compute the minimum hamming distance between all possible $r_{i,j}$ and the modeled response r'_i.

$$f_i = min_{j=1,..,2^k}\{HD(r_{i,j}, r'_i)\} \tag{7}$$

The fitness f of a PUF model is then simply given by the sum $f = \sum f_i$. The smaller f, the fitter the PUF model. However, when the PUF model accuracy is very low, it is likely that for the computation of f_i the wrong response candidate $r_{i,j}$ is used, i.e. $r_i \neq r_{i,j}$. In this case the fitness value f_i is misleading. However, the higher the model accuracy and the more inputs are used, the more likely it becomes that the correct PUF model is chosen as the fittest.

We have simulated the attack using a 64-bit Arbiter PUF and 7 inputs each consisting of 234-bit helper data corresponding to a 255 response string. Note that $q = 7$ is the default value of the reverse fuzzy protocol and hence a single execution of the authentication protocol reveals 7 inputs. To measure the model accuracy we used 10k challenge and responses as a reference set. The results of the attack are shown in Figure 2. ES-machine learning attacks are non-deterministic. Given the same input, the algorithm can lead to different results. From 100 tries with the same inputs, 24 tries were successful and achieved a model accuracy of at least 96% percent. In a second experiment we applied Gaussian noise to the delay values so that 5% of the responses flipped to test the impact of noise to our attack. The number of successful tries decreased only slightly from 24 runs without noise to 19 runs with 5% noise. A single run with 7 inputs took around 23 minutes using around 16 cores on a cluster.

From Figure 3 and 4 one can see that once a PUF model accuracy of more than about $\approx 60\%$ is achieved, the attack is successful and the model accuracy quickly increase to 98%. This is due to the fact that for small model accuracies the fitness value f is not very meaningful. Only for model accuracies above $\approx 60\%$ is the fitness value meaningful as can be seen in Figure 4. We also tested the attack against 128-bit Arbiter PUFs. While for small number of inputs the attack was not successful we were able to model a 128-bit Arbiter PUF with 200 inputs. From 15 runs, two achieved a model accuracy of more than 98% after 500 generations while the other runs did not converged. A single run with 500 generations took about 210 minutes. Hence, larger Arbiter PUFs increase the attack complexity but cannot prevent the machine learning attack.

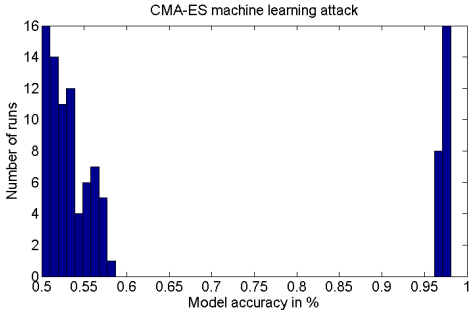

Fig. 2. Result of CMA-ES on the reverse fuzzy extractor using 7 inputs i.e. 7 syndromes on simulated responses from a 64-bit Arbiter PUF. The model accuracy was computed using 10k reference challenge and responses. 100 runs with the same challenges and PUF instance were performed from which 24 runs achieved a model accuracy of more than 96%.

D. ES-machine learning attack using noisy responses

The information for which challenges the PUF is unreliable, i.e. for which challenges the response might flip contains valuable information from an attackers perspective. Becker *et al.* [2] proposed a fault attack on controlled PUFs that uses the information which challenge bits are unstable under voltage variations to build an accurate PUF model. For this attack the attacker only needs to know which challenges are unstable, i.e., for which challenges the response might flip due to environmental or thermal noise. The response bits on the other hand are not needed for this attack to work. Delvaux *et al.* [5] used the amount of bit flips under thermal noise in addition to the response bits for a model building attack on an Arbiter PUF. The main observation in both attacks is that the

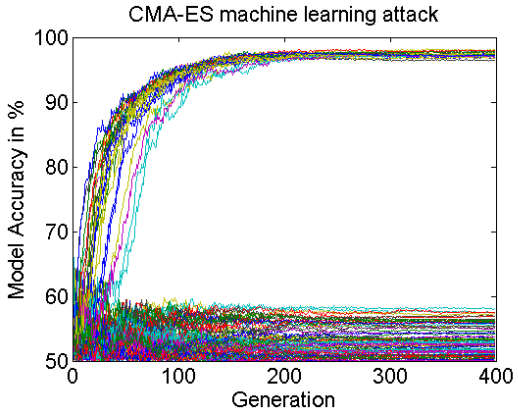

Fig. 3. Progression of 100 runs of the CMA-ES on the reverse fuzzy extractor with 7 inputs and a 64-bit Arbiter PUF. The Y-axis shows the achieved model accuracy after each generation.

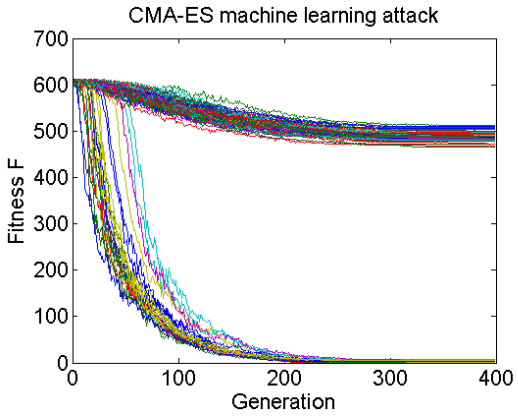

Fig. 4. Progression of 100 runs of the CMA-ES on the reverse fuzzy extractor with 7 inputs and a 64-bit Arbiter PUF. The Y-axis shows the computed fitness value f after each generation.

closer the delay difference for a given challenge is to zero, the more likely is it that the bit flips. On the other hand, the larger the delay difference, the less likely it is that a challenge bit flips. Figure 5 [2] shows the circuit-level simulated delay differences for different challenges of an 128-bit Arbiter PUF. The delay differences are approximately Gaussian and lie between roughly -100ps and +100ps. When the supply voltage is changed from the default 1.1V to 1V and 1.2V some of the challenges flipped, i.e. the sign of the delay difference changed. These challenges are highlighted in black. All flipped responses had a delay difference between -13ps and +13ps. Hence, every flipped bit gives us an important piece of information: the absolute delay difference for this challenge is very likely smaller than a threshold τ, where τ depends on the specific PUF instance (13 ps in this example).

It was demonstrated in [2] that an ES-based machine learning algorithm can be used to build an accurate model of an Arbiter PUF knowing only which responses are unstable. The question is, how do we determine the bits that are unreliable in the reverse Fuzzy extractor protocol? As it turns out, the BCH code hands us this information on a silver platter: For a response r and a noisy response r'

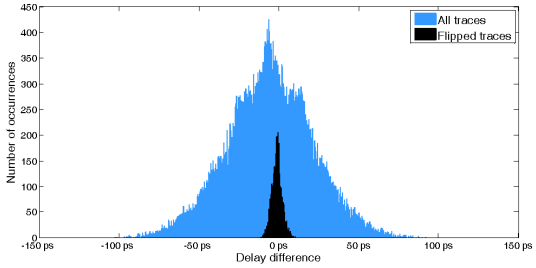

Fig. 5. The delay difference in pico seconds of an 128-bit Arbiter PUF for 49k different traces. Colored in blue are the delay differences of all traces and in black are the delay differences for the traces whose output flipped when the supply voltage was altered from 1.1V to 1V and 1.2V.

with $HD(r, r') < t$ and their corresponding public helper data h and h', one simply has to compute the error vector e from the reproduction function:

$$s = h - h' \tag{8a}$$

$$e = Decode(s) \tag{8b}$$

Since $e = r - r'$, the error vector tells us which bits have flipped and which bits were stable. This is exactly the information that is needed to perform a machine learning based fault attack.

The used ES-machine learning attack is very similar to the attack described in the previous Section. The main difference is the computation of the fitness test of the PUF models. The input of the fitness test is not the helper data h_i, but the error vector e_i that was computed from multiple helper data for the same challenge under different environmental conditions[1]. To evaluate the fitness of a PUF model, a modeled error vector e' is computed for every challenge. To do this, the delay difference for every challenge is computed and if the delay difference is below a threshold τ a bit flip is expected. The measured error vectors e_i are then correlated with these modeled error vectors e'_i and the corresponding correlation coefficient is used as a fitness indicator. Please note that the error vector e_i does not exactly match the modeled error vectors e'_i, since not all challenges whose delay value is below τ necessarily flipped during the measurement (see Figure 5). However, if the correct PUF model was used, the two error vectors should be *similar*. In our experiments the correlation coefficient worked very well to test this similarity.

One open question is how to set the threshold value τ, since this value depends on the PUF instance as well as the environmental conditions. A good solution is to simply add τ to the parameters that are to be determined by the ES-machine learning algorithm. By making it part of the machine learning parameters, the optimal value is determined by the algorithm on the fly and does not need to be chosen by the attacker.

We implemented this attack by assuming that Gaussian noise is added to the delay differences of each challenge.

[1]It is also possible to challenge the PUF using the same environmental conditions, since PUFs in practice also show some unreliability without changing environmental conditions.

We added Gaussian noise to the delay difference of each challenge so that 5% of the responses flipped, i.e. for 5% of the challenges the sign of the delay difference changed. The results of the attack for a 64-stage Arbiter PUF are depicted in Figure 6 and for a 128-stage Arbiter PUF in Figure 7. While the number of needed traces is slightly higher (14 input block instead of 7) compared to direct ES-machine learning attack, the needed number of inputs is still extremely small and the attack time is magnitudes faster. The biggest advantage of the attack however is that the attack is independent of the used BCH parameters as long as all errors are corrected.

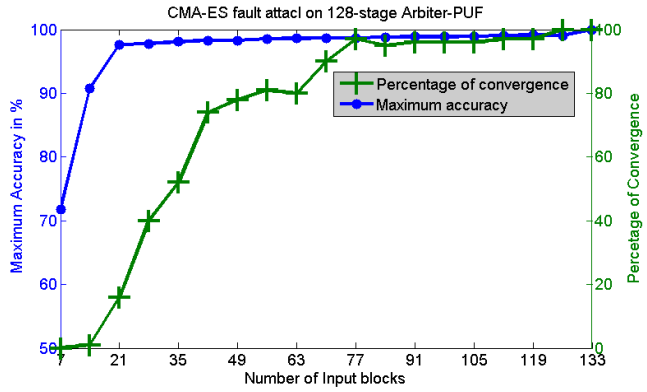

Fig. 6. CMA-ES attack on 64-stage Arbiter based on the information of which bits have flipped. The responses were generated by adding Gaussian noise to simulated delay values so that 5% of the responses flipped. On the left Y-axis, the highest achieved accuracy from 100 runs with 800 generations each are shown. On the right Y-axis the number of runs that converged, i.e. that achieved a model accuracy of at least 90% are shown. The X-Axis depicts the used number of input blocks, each input block consisting of 255 response bits.

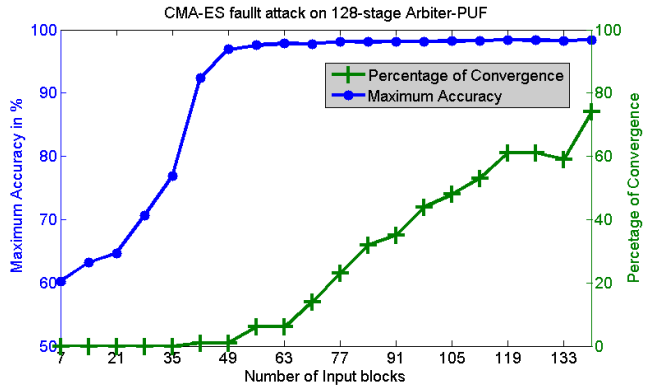

Fig. 7. CMA-ES attack on 128-stage Arbiter based on the information of which bits have flipped. The responses were generated by adding Gaussian noise to simulated delay values so that 5% of the responses flipped. On the left Y-axis, the highest achieved accuracy from 100 runs with 800 generations each are shown. On the right Y-axis the number of runs that converged, i.e. that achieved a model accuracy of at least 90% are shown. The X-Axis depicts the used number of input blocks, each input block consisting of 255 response bits.

This is a significant difference to the direct ES-machine learning attack on the helper data. In the previous attack, changing the parameters of the used BCH code and using a different PUF as a building block might be able to prevent the attack, since the attack complexity directly depends on the used BCH code. However, changing the parameters of the BCH code does not affect this attack. The information of which bits are unstable will always be leaked by a BCH code if the attacker can send the same challenge twice.

Hence, while the other machine learning attack might be prevented by using larger BCH codes and a harder to model PUF architecture, this attack exploits a fundamental weakness of BCH codes that cannot be easily fixed. The results presented in this section also have consequences beyond the reverse fuzzy extractor. Every protocol that uses BCH codes or other error correction codes that reveal information about which bits are unreliable need to carefully consider these fault attacks.

IV. Attacking the Slender PUF Protocol

The Slender PUF protocol was first introduced in [15], which we will refer to as the conference version and has also been published with small modifications in [17], which we will refer to as the journal version. The Slender PUF protocol is based on pattern matching, which has previously been proposed for correcting errors in PUF based key generation [16].

The protocol enables a token (called prover [15]) with physical access to the PUF to authenticate itself to a verifier. To do so, the verifier first builds an accurate model of the PUF during an initialization phase. During this initialization phase, the verifier received challenge and response pairs from the token which can be used to build an accurate PUF model using machine learning. After the verifier has built an accurate software model of the PUF, the initialization phase is permanently disabled and the token will never again directly reveal any challenge-and-response pairs. Instead, the token will only reveal a permuted substring of the response bits. The protocol of the conference version is shown in Table II. The protocol is started by the verifier by sending a nonce $nonce_v$ to the PUF. The token responds with a randomly generated nonce $nonce_t$. The two nonces are used as a seed for a pseudo random number generator (PRNG). This PRNG is then used in step 4 to generate L challenges C. The token computes the corresponding responses R using its physical PUF instance, i.e. R=PUF(C). In step 6 the token randomly chooses an index ind, with $1 \leq ind \leq L$, that points to a location of the response string R. The index is used to generate a substring W from the response string with a predefined length L_{sub}, with $W = R_{ind}, .., R_{ind+L_{sub}}$. The response string R is used in a circular manner, so that if $ind + L_{sub} > L$, $W = R_{ind}, .., R_L, R_1, ..R_{ind-L_{sub}}$.

This substring W is then sent to the verifier. The verifier computes its own response string R' using the software model of the PUF with R'=PUF_model(C). In the last step the verifier uses its computed response string R' to search for the index ind that points to W using a maximum-sequence alignment. Note that ideally $R = R'$ and hence the hamming distance between the transmitted substring W and the verifiers substring W'

is 0. However, in practice R and R' will differ slightly due to inaccuracies in the verifiers PUF model as well as noise in the physical PUF at the token side. Therefore the token accepts a few false response bits in the substring W. If the hamming distance between is below a certain threshold, $HD(W, W') < e$, then the authentication is successful. Otherwise the authentication fails and the protocol needs to be restarted.

TABLE II
CONFERENCE VERSION OF THE SLENDER PUF PROTOCOL [15]

Token physical PUF		**Verifier** PUF_{model}
	$\xrightarrow{nonce_v}$	
	$\xleftarrow{nonce_t}$	
$C = G(nonce_v, nonce_t)$		$C = G(nonce_v, nonce_t)$
$R = PUF(C)$		$R' = PUF_{model}(C)$
$W = SEL(ind, L_{sub}, R)$		
	$\xrightarrow{W}$	
		$T = match(R', W, e)$
		T=true?

TABLE III
JOURNAL VERSION OF THE SLENDER PUF PROTOCOL [17]

Token physical PUF		**Verifier** PUF_{model}
	$\xrightarrow{nonce_v}$	
	$\xleftarrow{nonce_t}$	
$C = G(nonce_v, nonce_t)$		$C = G(nonce_v, nonce_t)$
$R = PUF(C)$		$R' = PUF_{model}(C)$
$W = SEL(ind, L_{sub}, R)$		
$PW = PAD(ind_2, W)$		
	$\xrightarrow{PW}$	
		$T = match(R', PW, e)$
		T=true?

For the journal version [17] the protocol was modified slightly as can be seen in Table III. In the journal version of the protocol, the substring W is not directly revealed but a padding is applied to W. In the first step of the padding process, L_{pw} random padding bits are generated. Then a second random index ind_2 is chosen with $1 \leq ind_2 \leq L_{pw}$ and the string W is inserted into the padding bits at position ind_2. This process is illustrated in Figure 8. The padded substring PW is transmitted to the verifier. The verifier then performs a circular maximum-sequence alignment of the received string PW and its simulated PUF output sequence R'. This way the verifier can determine the substring W and the secret indices ind_1 and ind_2. If the hamming distance between the simulated substring W' and the received substring W is below a certain threshold, the authentication was successful.

The journal version has the disadvantage that the transmitted string PW is longer than W and the verifier has to do more computations to find the substring W. On the other hand, in the journal version two secret indices are used and [17] proposes to use these indices to extend the authentication protocol to a session key exchange protocol.

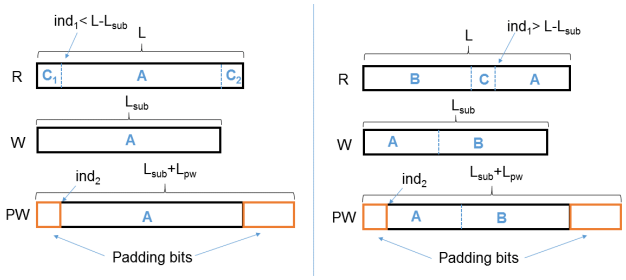

Fig. 8. Generation of the substring W and PW for the case $ind_1 < L - L_{sub}$ and for the case $ind_1 > L - L_{sub}$.

A. Security of the Slender PUF protocol

The security analysis of the Slender PUF protocol provided in relies on the fact that an attacker does not have access to direct challenge and response pairs. The Slender PUF Protocol uses XOR-Arbiter PUFs in which the responses of multiple arbiter PUFs are XORed to generate a single output bit [20]. XOR-Arbiter PUFs are known to be much more resistant against machine learning attacks than simple Arbiter PUFs [18]. However, the number of XORs that can be used is limited since the reliability decreases with each added XOR. In [17] a worst case error rate e of 24.7%, 34.6%, and 43.2% for an 2-input, 3-input and 4-input XOR Arbiter PUF with 64 stages respectively was measured in their FPGA prototype. But as also pointed out by Majzoobi *et al.* , much smaller error rates have been reported for ASIC implementations [12]. While modeling attacks on XOR Arbiter PUFs are more difficult than on single Arbiter PUFs, model building attacks are still possible if enough challenge and response pairs are known. For their reference FPGA implementation in [18], empirical results show that for the used 3-XOR Arbiter PUF 64k challenges and responses are needed for a successful machine learning attack. However, please note that this result is only valid for their PUF measurements with the very high error rate of 34.5%. If the used responses are more reliable much fewer challenges are needed to accurately model a 3-XOR Arbiter PUF. Figure 9 shows our result of an ES-machine learning on a 3-XOR arbiter PUF using simulated noise-free challenge and responses. From Figure 9 we can see that more than $N_{min} = 6000$ challenge and responses are needed to model a 3-input XOR-Arbiter PUF with the employed CMA-ES machine learning algorithm.

The main security argument why the Slender PUF protocol is resistant to machine learning attacks is that it would be computationally infeasible to compute enough direct challenge and response pairs to model the used XOR PUF [15], [17]. To get N_{min} challenge and responses, an attacker would need N_{min}/L_{sub} correct substrings W. To guess a substring W, an attacker has to guess the indices ind_1 and ind_2 correctly. For the proposed values of $L = 1300$, $L_{sub} = 1250$ and $L_{PW} = 512$ an attacker would

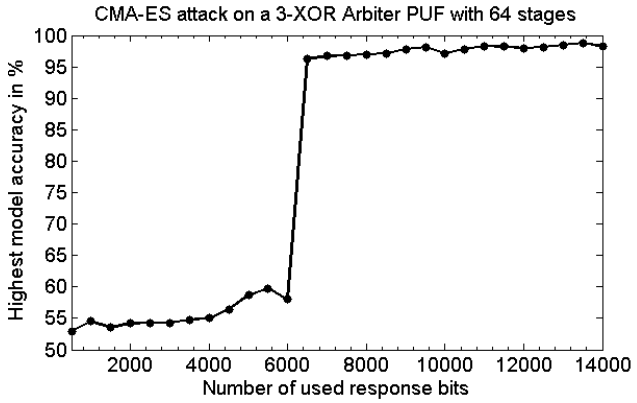

Fig. 9. CMA-ES attack on 3-Xor Arbiter PUF with 64-stages for different number of used responses. Noise-free simulated challenge and responses were used for this analysis.

need to perform

$$\begin{aligned} &O((L \times L_{PW})^{\lceil \frac{N_{min}}{L_{sub}} \rceil}) \\ &= O(1300 \times 512)^{\lceil \frac{6000}{1250} \rceil}) \approx O(2^{96}) \end{aligned} \tag{9}$$

machine learning attacks to attack the journal version of the protocol. Hence, according to this analysis, it would be computational infeasible to attack the protocol using a machine learning attack. Using the same logic, the attack complexity against the conference version of the protocol with $L = 1024$, $L_{sub} = 256$ would be:

$$O(L^{\lceil \frac{N_{min}}{L_{sub}} \rceil}) = O(1024^{\lceil \frac{6000}{256} \rceil}) \approx O(2^{230}) \tag{10}$$

However, we will show that a ES-based machine learning attack on these protocols is feasible. This is due to the fact that in an ES machine learning attack the attacker does not need to guess the correct indices ind_1 and ind_2. Instead of trying to guess the indices to get more than 6000 challenge and response pairs, the attacker can directly use the strings PW or W as inputs to a CMA-ES based machine learning attack.

B. ES-Machine Learning Attack on the Slender PUF protocol

As discussed in Section II-C, for a successful ES machine learning attack on a PUF design, an attacker needs to be able to model the underlying PUF and needs a fitness test that can determine which PUF instances from a given set of PUF instances are the fittest, i.e. which instances model the PUF the best. In the Slender PUF protocol, the challenges and responses are never directly revealed. Hence, an attacker cannot use the model accuracy as the fitness test. Let us first take a look at the conference version of the Slender PUF Protocol. In the conference version, the substring W of length L_{sub} is transmitted without applying any padding. To find the correct index ind_1, the verifier performs a maximum sequence alignment with his string R' and W. That is, the verifier computes for all possible indices ind_1 all substrings W' and computes the hamming distance $HD(W', W)$. The authentication passes if $min\{HD(W, W')\} < t$.

The attacker uses the same method for his fitness test. Assume the attacker has eavesdropped (or initialized) n executions of the protocol and collected n substrings $W_1, ..., W_n$ and their corresponding challenges $C_i, ..., C_n$. To test the fitness of a PUF instance generated during the ES-machine learning attack, the attacker computes responses $R'_1, .., R'_n$ with $R'_i = PUF_{model}(C_i)$. In the next step he performs a maximum sequence alignment of the computed responses R_i with the eavesdropped substrings W_i to find the minimum hamming distance for every substring. These minimum hamming distances are summed up and are used as a fitness metric f:

$$f = \sum_{i=1}^{N} min\{HD(R'_i, W_i)\}$$

A PUF instance with a high model accuracy will have a smaller hamming distance f than a PUF instance with a low model accuracy. When the model accuracy of a PUF instance is high, it is very likely that the minimal Hamming distance corresponds to the correct index. If the PUF model accuracy is very low, it is possible that the correct index might have a higher hamming distance than a false index. However, if enough strings W are used to evaluate the fitness of the PUF models, it is more likely that the PUF models with a higher model accuracy are chosen as parents for the next generation than PUF instances with a worse model accuracy. And as long as fitter instances have a higher chance to be used as parents ES-based machine learning can be used.

We performed an CMA-ES machine learning attack on the conference version of the Slender PUF protocol with the default parameter provided in [15], $l = 1024$ and $l_{sub} = 256$. For our attack we used Matlab together with C functions for the computationally expensive steps, the response generation and the determination of minimal hamming distance. We followed the general approach of assuming a Gaussian distribution of the delay parameters [18] and used this delay model to simulate the PUFs. We generated two random sets of challenges. One set of challenges was used to compute the responses R that are used to generate the strings W. The other set of 10k challenges was used to evaluate the model accuracy at the end of the attack. A CMA-ES run with 350 generations and 60k inputs (each input consisting of the 256-bit string W) took around 17 hours using 16 cores of a multi-core workstation. From 8 runs, one run achieved a model accuracy of 95%, one run only partially converged to 36.5% while the other runs did not converge and stayed close to 50%. Please note that each run was aborted after 350 generations due to the long execution time of the attack although the model accuracy of the successful run was still improving. But with a model accuracy of 95% an attacker can recover the secret indices and hence the direct challenge and responses. With this information a machine learning attack that achieved an accuracy grater than 99%

just takes a few seconds.

The results show that the Slender PUF protocol can be attacked if a 3-XOR Arbiter PUF is used and that the Formula 9 cannot be used to estimate the attack complexity of a machine learning attack. This is mainly due to the fact that the substring itself can be used as the input to the machine learning algorithm and the correct indices do not need to be guessed. Instead, the correct indices will be one of the outputs of the machine learning attack.

C. Attacking the journal version

In the journal version of the Slender PUF protocol a padded string PW is transmitted instead of W. In [17] the proposed parameters for a 3-input XOR Arbiter PUF are $L = 1300$, $L_{sub} = 1250$ and $L_{PW} = 512$. In principle, the same attack as described for the conference version of the protocol can be used to attack the journal version. However, in the journal version two secret indices are used to generate the string PW so that a verifier needs to compare $L \times L_{pw} = 1300 \times 512 = 665600$ different strings to determine the minimal hamming weight. For comparison, in the conference version only $L = 1024$ comparisons are needed. While in theory it is possible to test all 665600 substrings, doing so will considerably increase the computation time. Since computing the minimal hamming distance is the most time consuming operation in this machine learning attack, it is recommended to use a more efficient method.

The main idea is to reduce the number of comparisons at the cost of using stings with less bits (and hence less information). We know that a L_{sub} bit string W is present in PW. If we assume that $ind_1 < L - L_{sub}$ we can see from Figure 8 that the string W consists of two string A and B. The string A consists of the first $|A|$ bits of the response R and the string B consists of the last $|B|$ bits of response R. Since $|A| + |B| = L_{sub}$, this means that either the first $L_{sub}/2$ bits of R or the last $L_{sub}/2$ bits of R are present in W. Instead of trying to detect the entire string W in PW we only try to find either the first $L_{sub}/2$ or last $L_{sub}/2$ bits of R in W. To do this we only need $2 \times L_{PW}$ comparisons which reduces the computation complexity considerably.

However, we also have to consider the case that $ind_1 < L - L_{sub}$. If $ind_1 > L - L_{sub}$ we can see from the left side of Figure 8 that in this case the string W does not contain the first $|C_1|$ bits of R nor the last $|C_2|$ bits of R. But since $L - L_{sub} = |C_1| + |C_2| = 50$ this does not change much. We simply ignore the first and the last 25 bits of the response R for our analysis and only search for $L_{sub}/2 - 15 = 600$ bits. Using this methods we make $2 \times |PW| = 2 \times (L_{sub} + L_{pw}) = 3524$ hamming distance computations per input in our fitness test at the cost of only using 600 bits instead of 1250 bits of substring W. Compared to 665600 computations this is a significant speedup that makes the attack much more practical.

We used lazy evaluation for our attack, i.e. we only used a subset of the all inputs to evaluate in each generation. We used 200k inputs of which we used 60k in each generation. From 13 runs 1 runs were successful with an achieved model accuracy of 92% after 340 generations. The PUF model accuracy for this run was still increasing and would have reached a much higher level if the run would not have been stopped. However, just as was the case for the conference version, with such a high model accuracy an attacker can compute the secret indices and then directly use challenge and responses in a second machine learning algorithm. This is considerably faster than continuing the attack with the substring method. The computation time of 340 generations using ≈16 threats on a multi-core cluster took about 31 hours. However, the sigma value in conjunction with the fitness test can often indicate if an attack was successful or not and hence some unsuccessful tries can be aborted earlier.

The presented results show that it is possible to attack the Slender PUF protocol using machine learning attacks. The results indicate that the computation complexity is non-trivial, but far from computational infeasible. An attacker can model the PUF with moderate resources in reasonable time.

V. Conclusion

In this paper we have demonstrated at the example of the reverse fuzzy extractor and the Slender PUF protocol how powerful machine learning attacks can be. The main lesson learned is that machine learning attacks are possible even if no direct challenge and responses are available to an attacker. Access to highly obfuscated responses such as the substrings in the Slender PUF protocol or the helper data of error correction codes can be enough to perform an ES-based machine learning attack. A common approach to proof the security of a PUF protocol is to show that an attacker does not have access to a certain number of direct challenges and responses. This comes from the common believe that for a successful machine learning attack the attacker needs to know a certain number of challenges and responses. However, as demonstrated in this paper, direct challenge and responses are not always needed to perform machine learning attacks. Highly obfuscated responses can still be used to accurately model a PUF. Hence, the machine learning complexity of a PUF protocol is not the same as the complexity of computing a certain number of direct challenges and responses.

Furthermore, the attack on the reverse fuzzy extractor demonstrated how much valuable information helper data from error correction codes can contain. The important point is that these error correction codes do not necessarily need to leak the individual response bits to be useful for an attacker. The information which challenges are more robust than others can be used for a machine learning algorithm as well. This is especially problematic for error correction codes such as the BCH codes that directly reveal which bits have flipped if the same challenge is applied twice. But other error correction codes can contain similar information that can be exploited by an attacker. Hence, when choosing error correction codes for delay

TABLE IV
SUMMARY OF THE MACHINE LEARNING ATTACKS ON THE DIFFERENT PROTOCOLS.

Protocol	maximum accuracy	successful runs	used inputs	execution time per run
Reverse Fuzzy Extractor 64-bit	97%	24/100	7	≈ 23 minutes
Reverse Fuzzy Extractor 64-bit (Fault Attack)	97%	10/100	21	<1 minute
Reverse Fuzzy Extractor 128-bit	99%	2/15	200	≈ 210 minutes
Reverse Fuzzy Extractor 128-bit (Fault Attack)	97%	6/100	56	<1 minute
Slender conference version	95%	1/8	60k	≈ 20 hours
Slender journal version	92%	1/13	(60k)200k	≈ 31 hours

based PUFs, machine learning attacks need to be carefully considered.

REFERENCES

[1] F. Armknecht, R. Maes, A. Sadeghi, O.-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.

[2] G. T. Becker and R. Kumar. Active and passive side-channel attacks on delay based puf designs. *IACR Cryptology ePrint Archive*, 2014:287, 2014.

[3] X. Boyen. Reusable cryptographic fuzzy extractors. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 82–91. ACM, 2004.

[4] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede. Secure lightweight entity authentication with strong pufs: Mission impossible? In *To appear in Cryptographic Hardware and Embedded Systems (CHES 2014)*, 2014.

[5] J. Delvaux and I. Verbauwhede. Side channel modeling attacks on 65nm arbiter pufs exploiting cmos device noise. In *6th IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2013)*, June 2013.

[6] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in cryptology-Eurocrypt 2004*, pages 523–540. Springer, 2004.

[7] 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.

[8] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls. Fpga intrinsic pufs and their use for ip protection. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 63–80. Springer Berlin Heidelberg, 2007.

[9] N. Hansen. The cma evolution strategy: A comparing review. In *Towards a New Evolutionary Computation*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 75–102. Springer Berlin Heidelberg, 2006.

[10] A. Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann. Reverse fuzzy extractors: Enabling lightweight mutual authentication for puf-enabled rfids. In *Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 374–389. Springer Berlin Heidelberg, 2012.

[11] G. Hospodar, R. Maes, and I. Verbauwhede. Machine learning attacks on 65nm arbiter pufs: Accurate modeling poses strict bounds on usability. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 37–42. IEEE, 2012.

[12] S. Katzenbeisser, Ü. Koçabas, V. Rozic, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann. Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon. In *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 283–301. Springer Berlin Heidelberg, 2012.

[13] 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. IEEE, 2004.

[14] R. Maes, A. Van Herrewege, and I. Verbauwhede. Pufky: A fully functional puf-based cryptographic key generator. In *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 302–319. Springer Berlin Heidelberg, 2012.

[15] M. Majzoobi, M. Rostami, F. Koushanfar, D. 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.

[16] Z. Paral and S. Devadas. Reliable and efficient puf-based key generation using pattern matching. In *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pages 128–133. IEEE, 2011.

[17] M. Rostami, M. Majzoobi, F. Koushanfar, D. Wallach, and S. Devadas. Robust and reverse-engineering resilient puf authentication and key-exchange by substring matching. *Emerging Topics in Computing, IEEE Transactions on*, PP(99):1–1, 2014.

[18] 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.

[19] U. Ruhrmair, J. Solter, F. Sehnke, X. 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.

[20] G. 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.

[21] G. E. Suh, C. W. O'Donnell, I. Sachdev, and S. Devadas. Design and implementation of the aegis single-chip secure processor using physical random functions. In *ACM SIGARCH Computer Architecture News*, volume 33, pages 25–36. IEEE Computer Society, 2005.