CaPUF: Cascaded PUF Structure for Machine Learning Resiliency

Hassan Nassar, Lars Bauer, Jörg Henkel
Chair for Embedded Systems, Karlsruhe Institute of Technology
Karlsruhe, Germany
{hassan.nassar, lars.bauer, henkel}@kit.edu

***Abstract*—With the rise of the Internet of Things (IoT), resource-constrained and power-constrained devices attract more attention. The need for lightweight solutions as alternatives to resource-intensive applications became more urgent. Moreover, as the number of connected devices grew, authenticating them became more challenging. Traditionally, this would be performed by using hash functions and secure memory to store a key, which both come at a high cost.**

Physical Unclonable Functions (PUFs) emerged as a suitable lightweight alternative to hash functions to authenticate the devices. Using the inherent minute differences between Integrated Circuits (ICs), they can generate IC-specific responses for input challenges coming from a so-called verifier.

Through the years, Machine Learning (ML) has been used to attack PUFs by modeling them and accurately predicting their response to a given challenge. This stimulated research on ML-resilient PUFs. This resilience came with significant area and challenge-to-response delay overheads. In this work, we introduce the novel Cascaded PUF (CaPUF) and show that it is resilient against state-of-the-art ML-based attacks, i.e., Logistic Regression (LR) and Support Vector Machines (SVMs). These attacks could not achieve accuracy better than 52% against our CaPUF, which is only as good as flipping a coin. Additionally, our CaPUF requires 89% less area compared to state-of-the-art ML-resilient PUFs.

***Index Terms*—Hardware Security, Physical Unclonable Function, Modelling Attacks.**

I. Introduction

Since their introduction in 2002 [1], PUFs became a highly investigated topic for their interesting properties. They offer a lightweight alternative for cryptographic primitives by generating IC-specific signatures based on the process variation between ICs. The PUF design is not a secret, however, without having physical access to the IC that implements it, its behavior cannot be predicted. Different PUF designs exist, and their functional concept is the same. The PUF receives an input challenge and processes it to produce a response, resulting in a Challenge Response Pair (CRP) relation. PUFs can be used to store cryptographic keys in the IC or to have a DNA-like identity of the IC. Instead of explicitly storing such a key/identity in a secure memory, it is generated on demand, by using the PUF and a public challenge. Numerous works investigated the usage of PUFs for different applications, such as attestation [2], RFID tags [3], IoT [4], electronic transaction protocols [5], secure FPGA reconfiguration [6], and secure code execution [7]–[9]. As most of these applications are usually performed on resource-constrained ICs, the PUFs need to be as lightweight as possible [4], [5], [10], i.e., using as few resources as possible. A PUF is considered lightweight if it can be implemented using a few hundred logic gates [11], [12].

PUFs are categorized into two categories, i.e., weak PUFs and strong PUFs. Strong PUFs have an exponential number of CRPs, while weak PUFs have linear number of CRPs [13]. Weak PUFs are more suitable for cryptographic keys storage while strong PUFs are mostly used for authentication. Note that the terms 'strong' and 'weak' are not related to their attack resilience, but they denote the number of generated CRPs.

As PUFs are used in cryptography and security scenarios, attacks against them have been proposed. An attacker with physical access to the PUF, or eavesdropping on the communication channel containing the challenges and responses, can collect a large number of CRPs, which can then be used to build an ML model to predict responses for challenges [14], [15]. Therefore, new PUF designs to counteract the modeling attacks have been proposed. Many of them incorporate cryptography to secure the output and make it unbreakable by ML models. But this comes at high resource- and latency overhead to get the proper output [12], [16], which breaks the lightweight property of the PUF.

Alternatively, other works propose to perform structural modifications to the PUF. For example, the XORPUF xores the output of several internal PUFs [17]. This increases the complexity of the ML model building, as it has to accurately model all the internal PUFs. However, using several PUFs to produce one output incurs additional overhead and can lead to resource usage near the PUFs that incorporate cryptography. Another direction to protect PUFs is to limit their responses to a list of authenticated challenges [18], [19]. In case the PUF receives an unauthenticated challenge, it either locks and does not produce an output or produces an obfuscated response. This reduces the ability of the attacker to collect CRPs to model the PUF behavior.

In this work, **we propose the novel Cascaded PUF (CaPUF) design**, an ML-resilient lightweight PUF. In comparison to the state of the art, **our novel contributions are as follows:**

- Our novel Cascaded PUF (CaPUF) design is resilient

Manuscript received April 07, 2022; revised June 11, 2022; accepted July 05, 2022. This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2022 and appears as part of the ESWEEK-TCAD special issue.

against ML modeling attacks. It internally combines several very lightweight PUFs, such that CaPUF altogether remains lightweight. The internal PUFs are organized in stages and are connected in chains, such that the output from one PUF at each chain is used as input for the next stage. The output from the different chains controls the behavior of PUFs on the other chains. Based on this cascade, CaPUF is able to achieve ML-resilience.
- CaPUF is lightweight and requires 89% less area compared to state-of-the-art PUFs.
- CaPUF is capable of performing locking and obfuscation with an acceptable overhead in comparison to state-of-the-art PUFs and without leaking any data to an attacker.
- Additionally, to the best of our knowledge, our work is the first to report successful attacks against the Pseudo Linear Feedback Shift Register PUF (PLPUF) design.

The rest of the paper is structured as follows. In Section II we give the necessary background about PUFs and the attacks used against them. The related works to counteract these attacks are discussed in Section III. The design of our novel CaPUF is detailed in Section IV accompanied by a mathematical model showing the hardening against ML-based attacks. Section V explains how to implement obfuscation and locking for CaPUF. The results for CaPUF and the comparisons with state of the art are detailed in Section VI, and Section VII provides the conclusions.

II. Background

PUFs harness the random deviations in the manufacturing process of ICs. Such deviations are unpredictable and cannot be avoided. Hence the output from a PUF cannot be predicted, even by the PUF designer, without physical access to the IC itself. Most PUFs are designed for digital ICs, but also other types of PUFs exist, such as optical PUFs [20], analog PUFs [21], mechanical PUFs [22], [23], and quantum PUFs [24].

In this work, we focus on delay-based digital PUFs [25], [26], which is one of the two main categories of digital PUFs beside memory-based digital PUFs [27], [28]. One common characteristic of PUFs is that they are noisy, i.e., some bits of the response might flip from time to time. To combat this, fuzzy extractors and error correction methods are applied to extract the response [2], [29]. These methods can either be applied on the IC containing the PUF or on the verifier side, which sends the challenge and receives the response. Not all PUFs have the same noise level. Some of them have a high noise level and are considered unreliable, whereas others have a low noise level and are thus considered reliable.

A. PUF Quality Metrics

The reliability against noise is one of the three basic metrics used to evaluate PUFs. The other two metrics are uniformity and uniqueness. Each metric has an ideal value as we discuss in the following. If the metrics of a PUF are near the ideal value, then the PUF is said to have good performance, otherwise, it has bad performance. The reliability is evaluated based on

$$\text{Reliability} = (1 - \frac{1}{N}\sum_{i=0}^{N-1}\frac{HD(R_s, R_i)}{m}) \times 100\%, \quad (1)$$

where N is the total number of measurements used to calculate the reliability, m is the bit-length of the response generated by the PUF, R_s is the stable response of the PUF at normal conditions, R_i is the response of the i-th measurement, and HD is the Hamming distance between two responses, i.e., the number of different bits. Ideally, the reliability should be at 100%.

The uniformity metric measures the frequency of 1s in the response, i.e., its Hamming weight. The probability of 0 and 1 should be equal, i.e., ideally, the uniformity should be at 50%. Uniformity is calculated by Eq. (2), where $R(i)$ is i-th bit of a response binary string.

$$\text{Uniformity} = \frac{1}{m}\sum_{i=0}^{m-1} R(i) \times 100\% \quad (2)$$

The uniqueness metric measures how unique a PUF is compared to other PUFs. If PUF-responses are similar across different ICs, it means that the PUF design is not governed by the process variations but rather by the delay paths of the design itself. Ideal uniqueness should be at 50%. If the uniqueness is higher, this would mean that the responses are similar. A lower uniqueness would mean that the bits are also similar but with inverted values. Uniqueness is important, as an attacker might have a reference PUF at hand. If the uniqueness is bad, then the attacker can easily model the PUF based on the reference PUF. Uniqueness is calculated based on Eq. (3), where Z is the number of PUFs, P_i is the response of the i-th PUF, and P_j is the response of the j-th PUF.

$$\text{Uniqueness} = \frac{2}{Z(Z-1)}\sum_{i=0}^{Z-2}\sum_{j=i+1}^{Z-1}\frac{HD(P_i, P_j)}{m} \times 100\% \quad (3)$$

B. PUF Usage

As mentioned in Section I, PUFs are considered weak or strong, based on their ability to produce Challenge Response Pairs (CRPs). The two delay-based PUFs used extensively in the literature are the Ring-Oscillator PUF (ROPUF) for weak PUFs [25], [30] and the Arbiter PUF (APUF) for strong PUFs [14], [31]. Weak PUFs are usually used as a Physically Obfuscated Key (POK). They are usually fine-tuned to have high reliability (around 99%) for one challenge. Note that even if the challenge is known, the key itself (i.e., the PUF response) cannot be modeled as each bit can either be 0 or 1, and usually the key is not communicated to the outside of the IC.

Strong PUFs can also be used as POKs, as one can fine-tune the response for one challenge to have high reliability. However, as they have a large set of CRPs, they are more often used for device authentication. After the manufacturing, the manufacturer or a trusted third party performs a so-called enrollment operation in which the challenges are sent to the

PUF and the responses are collected, which forms a golden list of CRPs. After deployment, the verifier uses this list to verify the identity of the device. By sending one challenge from the list and comparing the response, the verifier can check whether or not it matches the expected value. Weak PUFs are not suitable for device authentication. Due to their reduced set of CRPs, the challenges have to be repeated after some time, which makes them vulnerable to replay attacks. For example, when using CHOICE [32], then only 6,144 CRPs are possible. If CHOICE would be used for device authentication, then the device can be authenticated at most 6,144 times with a unique key that was never used before. Afterward, the challenges have to be reused, which means that a potential attacker might have eavesdropped on the correct response from earlier usage.

C. ML-modeling Attacks

Avoiding replay attacks comes with a new vulnerability, as it allows ML modeling attacks. With each authentication of the PUF, an attacker can get access to a new CRP. Using a large-enough set of CRPs, the attacker can effectively model the PUF.

Delay-based PUFs are usually built as a chain of several stages. The different delays of the different stages can be modeled with an additive linear model [14] for many PUFs. This makes it easy to attack the PUF with only a few thousand CRPs [14], [31]. The modeling is typically done using Logistic Regression (LR) or Support Vector Machines (SVMs) [16], [18]. APUFs, ROPUF, bistable ring PUF (BiPUF), and the dual-mode PUF are among the PUFs that can be modeled [31]. The ease of modeling a PUF is related to the simplicity of its mathematical model [14], [33]. For example, it has been shown that a simple constant but unknown rotation on the input challenge makes the PUF as hard to model as if it had quadratically many challenge bits (e.g., if it has 16 bits, then it is as hard as if it had 256 bits) [34]. Note that memory-based PUFs can also be modelled using ML attacks [27]. However, in this work, we focus on delay-based PUFs.

ML-modeling is not the only attack that targets PUFs. Other attacks against PUFs exist such as side-channel attacks [13], [35], but they are not in the scope of this work.

D. Pseudo Linear Feedback Shift Register PUF (PLPUF)

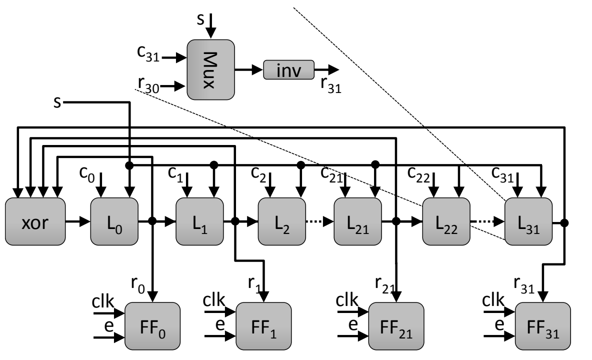


Fig. 1: The design of the PLPUF with a 32 bit bitwidth

As we detail in Section IV, the PLPUF [36] is the building block of our novel CaPUF. Therefore, we briefly describe its internal design in the following. The PLPUF is based on an LFSR, but it uses combinational elements instead of sequential registers. Figure 1 shows the design of a PLPUF for 32-bit challenge and response. It consists of 32 elements L_i, each of which contains an inverter that is connected to a multiplexer. Based on a select signal, either the initial challenge value or the output from the previous element is passed through the multiplexer to the inverter. The input for L_0 comes from an xor gate whose inputs are coming from the same elements as the xor inputs of a same-sized LFSR. The responses r_i are the outputs of the elements L_i, and they are stored in Flip-Flops FF_i. Using an enable signal, the Flip-Flops store the response exactly one clock cycle after the PLPUF is initialized by the challenge. Besides the input challenge, the stored response depends on the L_i latencies and the clock frequency. The frequency is constant and typically known, but the L_i latencies depend on IC process variation. As we show in Section VI-C, with enough CRPs, the ML-model can predict the responses. To the best of our knowledge, our work is the first to report the successful attacks against PLPUF.

III. Related Work

Several previous works tackled the issue of modeling attacks against PUFs. Here we explain the current state of the art. In Section VI-D we compare our work with all of them. Two main methods are used to achieve ML resilience. The first is cryptography-based and the other is architecture-based. The cryptography-based method incorporates cryptography, e.g., hash calculation, in the response evaluation. For example, LPN uses a Physically Obfuscated Key (POK) and the POK output and the challenge are concatenated into a string of bits. The hash of the concatenated string is the output response. LPN requires costly error correction [16]. Without it, the slightest error in the output of the POK would result in a totally different response.

FSM [19] and IPA [11] operate similar to LPN. However, they do not use a POK but a strong PUF. Moreover, FSM and IPA introduce ‘locking’ to the PUF, i.e., only a pre-approved challenge results in a valid output. Hence, the ability of an attacker to send challenges to collect responses is significantly limited. FSM and IPA achieve locking using different methods. FSM uses the strong PUF response as an input to a finite state machine. The response is divided int sub responses. Based on the values of the sub responses either it reaches the state that enables the hash calculation or resets. If it was a pre-approved challenge, then FSM continues to hash this response and this hashed value is the response. The locking mechanism of IPA is based on Fisher-Yates stream authentication [11] of the input challenge. If the challenge is authenticated, then the PUF works; if not, then it is locked. The final response is the hash of the response of the strong PUF. IPA improves over LPN by using pattern matching [27] and serial hash, but this comes with a significant latency penalty.

CRC [12] takes a different approach. It uses a stream cipher to change the polynomial of an LFSR that shifts the challenge before feeding it to an arbiter PUF. This breaks the linear dependency between the challenge and the response, and it improves the security by a quadratic factor [34]. That means: if the PUF uses an m bit long challenge, it requires the same effort to successfully model the PUF as if it had an m^2 bit long challenge. As the output comes from the PUF itself, there is no need to perform error correction. The verifier will be able to extract the response using fuzzy extractors. Nevertheless, the required stream cipher has a significant hardware overhead.

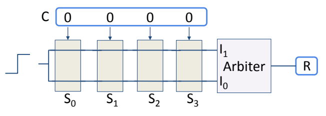
Another design using cryptography is the Slender PUF. It uses a True Random Number Generator (TRNG) to generate a random sequence of bits and the challenge is concatenated to this sequence. The concatenated string is used as input to a Pseudo Random Number Generator (PRNG) to generate the final challenge [37]. Based on secure communication of the output of the TRNG (e.g., encryption), the challenge can be rebuilt on the other end and it can be checked that it matches the response. Similar to CRC, there is no need to perform error correction on the device. However, the required TRNG comes with significant hardware overhead.

As for the architecture-based ML resilience, the design of the PUF is modified to complicate the mathematical model of the PUF even further. The earliest ML-resilient design is the XORPUF [17]. The output of several PUFs is xored. Thus, the output depends on several PUFs instead of a single one, which leads to a more complex model. However, it is not fully secure, as it can be seen as an addition of several linear models [15]. Moreover, for a small number of xored PUFs, one of them may have a higher effect on the final output than the others, leading to an easier attack [33], [34]. Hence, to make it secure, a high number of PUFs should be used, which incurs significant overhead.

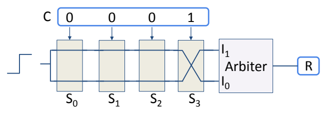
The CT PUF [31] uses three PUF designs together. Based on the number of the $1s$ in the even and odd positions in the challenge, the response is the output from either an ROPUF, an APUF, or a BiPUF. This further complicates the model and makes it harder, as it is not easy to build a full model for any of these PUFs. CT PUF introduces an optional extra layer of security by having the response as the output of an xor of two of the PUFs instead of the direct response from one PUF. This is similar to the XORPUF, just that different PUF designs are combined.

NoPUF [18] introduces obfuscation. It hides a reliable PUF in a noisy PUF, i.e, the PUF is only reliable for a subset of the challenges. NoPUF is based on the APUF design. NoPUF is only fine-tuned to be reliable for a certain subset of challenges. Similar to the locking of FSM and IPA, if the input challenge is not from that subset, then the output is noisy. This further complicates the modeling of the PUF, as the collected responses are unreliable. However, an attacker with knowledge of the NoPUF design can build a model based on reliability information [14], [15], [33]. Note that NoPUF is the one that shows similar overhead to our CaPUF, which is why we discuss its vulnerability in detail in the following.

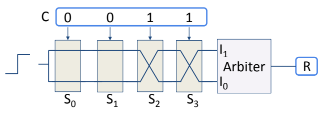
To further understand the NoPUF's vulnerability, Fig. 2 shows an APUF with four different challenges as examples. The APUF has two paths, upper and lower. A signal change from '0' to '1' is applied to both paths simultaneously and



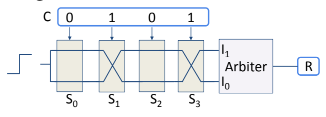
(a) APUF Stable with Challenge 0



(b) APUF Unstable with Challenge 1



(c) APUF Stable with Challenge 3



(d) APUF Unstable with Challenge 5

Fig. 2: Arbiter PUF used in NoPUF where some challenges lead to stable responses and other lead to unstable responses (the left-most bit of the challenge is the MSB)

traverses through the stages. At the output, an arbiter senses whether the signal change first arrived via the upper path (response bit '1') or lower path (response bit '0'). The APUF from Fig. 2 has four stages (S_0 till S_3) and each stage is controlled using one bit of the challenge. At each stage, if the challenge bit is '1', the two paths are switched.

The attacker would use the same challenge several times to get the reliability information. In this example, the attacker would send the challenge 0 several times and find that it is stable. Then the attacker changes only the least significant bit and sends challenge 1 several times and finds it to be unstable. From this difference, the attacker can deduce that switching the paths at S_3 causes both paths to have very similar delays. Then the attacker does the same using challenge 3 and challenge 5. For challenge 3, the PUF becomes stable again. Hence, switching the paths at S_2 makes one path significantly slower than the other. For challenge 5, the PUF remains unstable. Thus, switching the paths at S_1 cannot compensate for the instability caused by switching the paths at S_3. This valuable information enables the attacker to fine-tune the challenges sent to the PUF to get the best information about the different stages and build an accurate model.

IV. Cascaded PUF (CaPUF) Design

Our goal is to design a PUF that is resilient against ML modeling attacks. In our scenario, the PUF is used to perform device authentication, which is a common application of PUFs [2], [5]. The attacker collects Challenge Response Pairs (CRPs) either by eavesdropping on the communication channel or by impersonating the verifier. We aim to have a PUF with a complex delay model that is hard to model by using ML-based attacks, even when a large set of CRPs is collected.

A. Our Novel PUF Design

To achieve the ML resilience, we introduce our novel CaPUF. The main idea is to use several PUFs as building blocks to have a dynamically changing behavior from the PUF. With each new challenge, some of the properties of CaPUF change as if the challenge is given to a different PUF than with the previous one. CaPUF achieves the dynamic behavior through two aspects. The first aspect is the cascade. There are four stages of the PUF cascaded one after the other, the output from one stage is the input for the next stage. Hence, the response comes from an internal challenge that is different than the one sent initially. The PUFs used for the cascade are PLPUFs. We chose to use them as they are very lightweight PUFs.

The second aspect is the frequency to collect the outputs from the PLPUF. Instead of using a constant frequency all the time for all PLPUFs, for each stage, the frequency of collecting each individual response bit is variable. It depends on the output from another PLPUF from the previous stage. This introduces an extra layer of protection, as for each challenge the frequency of collecting each bit is different and the model of the PLPUF changes dynamically.

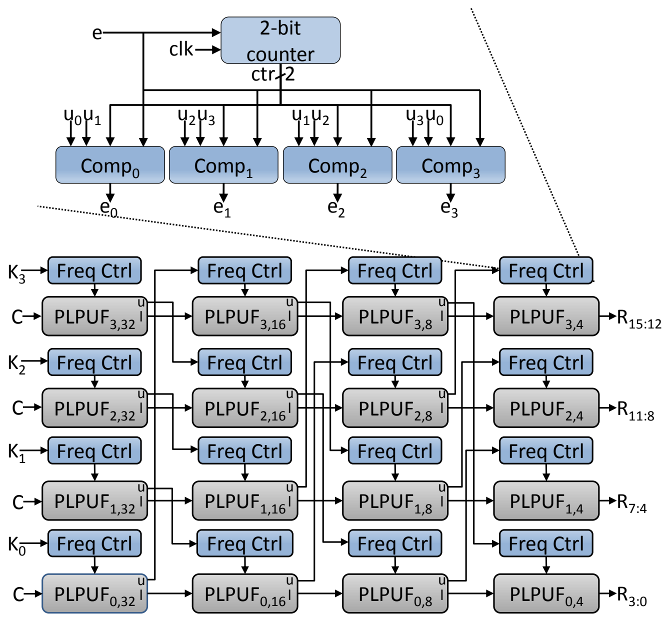


Fig. 3: Our Novel Cascaded PUF (CaPUF) containing 16 PLPUFs in a 4×4 grid. C is the 32 bit challenge sent to CaPUF, R is the 16 bit response (each 4 bits come from one PLPUF), and K_0 till K_3 are four secret keys.

Figure 3 shows the design of CaPUF. It has four stages (shown as columns in the figure) and each stage contains four PLPUFs. The challenges and responses of the PLPUFs of the first stage are 32 bit long (same as the design from Fig. 1), and the second/third/fourth stage use 16/8/4 bit, respectively. CaPUF has four chains (shown as rows in the figure) from chain$_0$ (bottom) till chain$_3$ (top). The output from one PLPUF in the chain is connected as an input to the next PLPUF in the chain. The 32 bit challenge is fed to all four PLPUFs of the first stage and the outputs of each PLPUF of the fourth stage are concatenated to form the 16 bit response of CaPUF. Each stage produces double the amount of bits needed as input for the next stage. The bits are separated into two signals, upper and lower, for all stages except for the final one. The lower part is used as an input for the next stage.

The upper part of the output is used to control the frequencies of the PLPUFs from the next stage. It does not control the same PLPUF that takes the lower part as input challenge. This choice is done such that no dependency between the PLPUF input and its frequency controller exists. The same chain controls the frequency on another chain only once to break any possible dependency between the chains. The frequency control of the first stage has to be initialized by a secret key (see the K_i values in Fig. 3). We use a POK to generate the K_i values. The POK can be any PUF with high reliability. For our case, we use a PLPUF with an arbitrary constant challenge to keep it lightweight.

In Fig. 3 we show the details of the frequency controller of the last stage, the other controllers look the same but with more comparators as they have more bits. The PLPUF is enabled for four clock cycles. A counter is used to track the clock cycles. Two bits from its input (the upper signal from a PLPUF of the previous stage) control the frequency of collecting one response bit. If the value from the two bits is equal to the counter, the enable signal for the Flip-Flop is turned on to collect the response bit. It is collected either after 1, 2, 3, or 4 clock cycles. The control is done by a simple 2 bit comparator. This makes each bit independent from the other.

While we use 16 PLPUFs for our CaPUF, it is still lightweight. Compared to the 32×16 APUF normally used to get 16 bit response from a 32 bit challenge, we use significantly less hardware and have ML-resilience. Further discussion of the overhead is presented in Section VI-D. We further evaluate alternative implementations with different number of stages in Section VI-E.

B. Complexity of Modeling CaPUF

To further understand why it would be hard to model CaPUF in the scope of a successful attack, we show an approximate mathematical model for its delay. The goal is to show that the mathematical model is more complex than other PUFs (e.g., APUF) and thus it will be harder to build an accurate model predicting the responses correctly as explained in Section II-C. Note that it is not the exact model but an approximated model for simplicity. The exact model would be significantly more complicated and even harder to be built through ML. Note that we don't rely only on the mathematical model, but we also run state-of-the-art attacks against CaPUF and the results can be found in Section VI-C. We start by modeling PLPUF (see Fig. 1), as it is the building block of our CaPUF. Based on this model we build the model for our CaPUF and show how in comparison it is a more complex model.

1) Mathematical Model of PLPUF: We illustrate the model for PLPUF from Fig. 1 with a bit width of 32 bits. We assume that all its L_i elements and the XOR-gate have the same delay D, which makes this an approximate model, as in practice the different L_i elements will have slightly different delays. Based on the period T of the clock signal ('clk' in Fig. 1), each bit c_i of a new input challenge will traverse through a constant number n of L_i elements

$$n = \frac{T}{D}. \tag{4}$$

This means that PLPUF will calculate n responses (all 32 r_i signals in Fig. 1 correspond to one response) for every input challenge c, but only the the n^{th} response will be sampled by the output register. In the following, we will use r^j to differentiate the n responses, and we will use r_i^j to denote bit i of the j^{th} response. The first response r^0 corresponds to the inversion of the challenge. All following responses r^j can be calculated as shown in Eq. (5). The first bit r_0^j of the j^{th} response comes from the xor gate, and the other bits come from the previous bit position from the $j-1^{\text{th}}$ response.

$$\begin{aligned} r_i^0 &= \overline{c_i} \quad i \in [0,31] \\ r_0^j &= \overline{r_0^{j-1} \oplus r_1^{j-1} \oplus r_{21}^{j-1} \oplus r_{31}^{j-1}} \quad j \in [1, n-1] \\ r_i^j &= \overline{r_{i-1}^{j-1}} \quad i \in [1,31], j \in [1, n-1] \end{aligned} \tag{5}$$

With enough CRPs, the model can deduce n and the number of transformations that occur through the PLPUF. The final model of a PLPUF that creates a response r can be expressed as an arbitrary function f based on n and the challenge c,

$$r = f(n, c). \tag{6}$$

f calculates the above-described shift and xor operations as shown in Eq. (5). This is a simplified version of the model. The real model would be a function of the delays of all L_i elements and the xor gate, which would all have different values. However, it is still a simple additive model that can be modeled by an ML-based attack after enough CRPs are collected.

2) Mathematical Model of CaPUF: Based on the approximate model for PLPUF, we can build the approximate model of our CaPUF. For illustration, we will only explain chain$_0$ from Fig. 3, but the same concept holds for all chains. We go step by step from the first stage to the final stage. Thus, showing how each stage in combination with the frequency controllers contributes to the model complexity which in turn makes CaPUF harder to model. We use the same subscripts as Fig. 3 which are two numbers separated by a comma. The first number is the chain identifier the second number is the bit width used for challenges and responses for the respective PLPUF on the chain. Starting by the first stage, the output from PLPUF$_{0,32}$ will be $r_{0,32} = f(n_{0,32}, c)$. $n_{0,32}$ can be computed as

$$n_{0,32} = \frac{K_{0,32} T}{D_{0,32}}, \tag{7}$$

where $K_{0,32}$ comes from the POK that stores the initial value for the first stage PUF, $D_{0,32}$ is the delay of one element in PLPUF$_{0,32}$, and T is the clock period. Here, the first effect of our design can be noticed. The response is collected after multiple clock cycles based on the secret value $K_{0,32}$. However, this alone would not be sufficient to prevent ML-based attacks, as $K_{0,32}$ is constant and the model will be able to learn it. That is why the other stages are added. The final output $r_{0,32}$ for the initial challenge c is

$$r_{0,32} = f(\frac{K_{0,32} T}{D_{0,32}}, c). \tag{8}$$

Going to the next stage in the chain, the input challenge for PLPUF$_{0,16}$ is the response from the previous stage $c_{0,16} = r_{0,32}$. $n_{0,16}$ is calculated as

$$n_{0,16} = \frac{r_{1,32} T}{D_{0,16}}, \tag{9}$$

where $r_{1,32}$ is the response from PLPUF$_{1,32}$ from Fig. 3. $r_{1,32}$ is variable and changes with each new input challenge. Hence, $n_{0,16}$ also changes with each challenge, which achieves the dynamic behavior that is desired for our design.

By substituting the values of $c_{0,16}$ and $n_{0,16}$ in Eq. (6), the final output from the PUF can be expressed as

$$r_{0,16} = f(\frac{f(n_{1,32}, c_{1,32}) T}{D_{0,16}}, f(n_{0,32}, c)). \tag{10}$$

Similarly, going to the third stage in the chain, the challenge $c_{0,8}$ is the output from Eq. (10). The value $n_{0,8}$ is evaluated as

$$n_{0,8} = \frac{r_{2,16} T}{D_{0,8}}. \tag{11}$$

$r_{2,16}$ is the response of PLPUF$_{2,16}$. Hence, $n_{0,8}$ is variable and gets a different value based on each new challenge.

By substituting $c_{0,8}$ and $n_{0,8}$ in Eq. (6) the response $r_{0,8}$ from this stage is based on

$$r_{0,8} = f(\frac{f(n_{2,16}, c_{2,16}) T}{D_{0,8}}, f(\frac{f(n_{1,32}, c_{1,32}) T}{D_{0,16}}, f(n_{0,32}, c))). \tag{12}$$

At the final stage, $n_{0,4}$ depends on $r_{3,8}$ and is formulated as

$$n_{0,4} = \frac{r_{3,8} T}{D_{0,4}}. \tag{13}$$

$r_{3,8}$ is based on a complex path similar to $r_{0,8}$. This makes $n_{0,4}$ variable and unpredictable as each PUF in the path of $r_{3,8}$ will affect its value. The final output from this chain is then based on

$$r_{0,4} = f(\frac{r_{3,8} T}{D_{0,4}}, f(\frac{r_{2,16} T}{D_{0,8}}, f(\frac{f(n_{1,32}, c_{1,32}) T}{D_{0,16}}, f(n_{0,32}, c)))). \tag{14}$$

It can be seen that the output is based on several outputs of PUFs from the cascade. The relation between the different PUFs is not a simple additive one but more complicated with multiplications.

To model the full CaPUF, the attacker would need to model several different PUFs, where their behaviors affect each other in multiplicative and transformations of inputs to outputs. Note that Eq. (12) and (14) are not fully expanded for brevity. The underlying model is even more complex. Moreover, this is just the approximate model. The real model will have a more complex structure. For once, the delay of all elements within one PUF is not equal. Thus, the calculation of n, the intermediate responses r^j, and consequently r will be more complex. Modeling all these PUFs and their relations will need an extremely high number of CRPs which might not be feasible to collect.

C. Knowledgeable Attacker

While the previous mathematical modeling shows the complexity of modeling CaPUF, an attacker with knowledge of the CaPUF design would try a cleverer attack. The attacker can either choose to attack the first stage or the last stage. Once it is successful in breaking one stage it will move forward (or backward) to break further stages.

The first alternative is that the attacker would try to break the first stage. This might seem reasonable, as the challenge for this stage is known. Moreover, even if the value K controlling the frequency of this stage is unknown, it is nevertheless constant and does not change its value from one challenge to another.

However, the attacker will not be able to find out the immediate response from this PUF stage. The output of this stage is used as input to the next stage and controls its frequency. The output is further transformed by the two intermediate stages and the output from the final stage has no direct relation to the initial challenge nor the output from the first stage. Hence, the attacker will not gain any useful information to build the model of the first stage.

The other alternative for the attacker is to attack the final stage. The reason for this is that this is the stage from which the direct response comes. As the outputs are known, the matching degree between the model and the actual PUF can be measured.

But the attacker will also fail, as this stage has several unknowns. First, its input challenge is unknown, i.e., the attacker needs to model the PUF behavior without the knowledge of the challenge. The attacker also does not know the frequency control of the PUF for this challenge. Hence, the attacker will have to guess the challenge and its frequency control. The model might seem accurate based on a false assumption of the challenge and the frequency control. Therefore, when predicting the response to the other challenges, it will deviate and will not be accurate.

Finally, the attacker might try to break both the first and last stage together. Such a scenario would have been possible if we would have used only two stages. The attacker would build assumptions on the first and last stage. Step by step the model would get more accurate as the first stage has a direct relation to the last stage. However, the two intermediate stages of CaPUF act as transformation stages. Thus, breaking this direct relation makes the modeling infeasible. The two intermediate stages can be seen as a layer of protection against the modeling by a knowledgeable attacker.

D. CaPUF Generalization

We chose to implement CaPUF using PLPUFs as the building block. However, the concept of CaPUF can be used with any PUF as the building block. The two aspects (cascade and dynamic behavior) have to be fulfilled. The cascade aspect can be easily implemented by chaining PUFs. The dynamic behavior has to be chosen to be suitable with each PUF, e.g., the frequency control is not effective with APUF.

Fig. 4 shows an example implementation of CaPUF using APUF. It is a simplified version with a 2×2 grid instead of 4×4 for simplicity and explanation only. The cascade is implemented by having the two chains chain$_0$ and chain$_1$. The dynamic behavior comes from using a rotator. Based on the value of the upper signal, the lower signal will be rotated n times. As this upper signal changes with each new challenge, the rotation will have a different n value with each challenge. We evaluate CaPUF using APUF in Section VI-E and give it the name of CaPUF$_{AP}$.

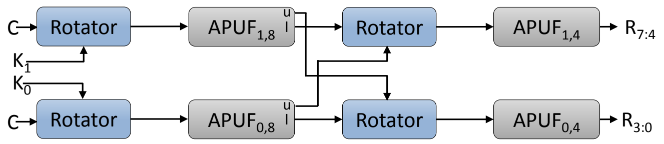


Fig. 4: Example for generalizing CaPUF by using APUF instead of PLPUF

V. CaPUF's Challenge Authentication

As mentioned in Section III, some PUFs are capable of locking or obfuscating their output if they receive an unauthenticated challenge. This limits the attacker's capability of collecting CRPs when impersonating the verifier. We now show how our CaPUF can be extended to perform obfuscation or locking.

First, to perform locking, we use a similar approach as used for FSM [19] (see Section III). However, our approach is more lightweight as instead of using a hash function, as done with FSM, we use an extra PUF.

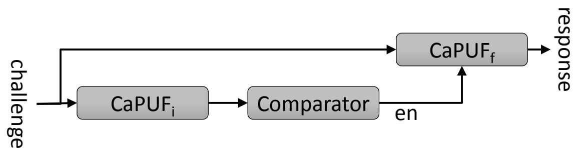


Fig. 5: Locking Mode for CaPUF

Fig. 5 shows the design of our locking-enhanced CaPUF$_L$. The challenge is used as an input to an initial CaPUF (CaPUF$_i$) and a final CaPUF (CaPUF$_f$) which are normal CaPUFs. The response comes from CaPUF$_f$ if it is enabled based on a signal from the comparator, else a constant string of 0s is the output. The comparator takes the output of the CaPUF$_i$ as an input and it performs the locking based on the value of the least significant 4 bits. If these four bits are equal to 1 or 2, then CaPUF$_f$ is enabled, otherwise it is disabled. The output of CaPUF$_i$ is equally distributed, the chance of having the four bits equal to 1 or 2 is 12.5%. We do not only assume that the output is equally distributed but we measured it for CaPUF$_i$ in our prototype and the probability is equal to 12.32% which is very close to 12.5%. This still leads to a large set of authenticated challenges; roughly 500 million for 32 bit challenges. During the enrollment phase (see Section II), the challenges that end up with a valid response are registered as authenticated challenges and will be used to authenticate CaPUF$_L$.

The attacker will then have a harder time collecting CRPs when impersonating the verifier, as there is no way to know which challenges will get a valid response. Moreover, the attacker cannot model CaPUF$_i$ as it is ML-resilient. Hence, CaPUF$_L$ will be able to perform locking with similar quality to FSM without the high overhead of using the hash function as CaPUF$_i$ comes with minimal overhead. This is further evaluated and discussed in Section VI. To lower the overhead even more, CaPUF$_i$ could have been eliminated and instead of using two CaPUFs we use a single modified CaPUF with a 5×4 grid instead of a 4×4 grid, which provides the extra 4 bits needed for the locking decision. The reason we did not implement this is that it would leak data from the modified CaPUF itself. CaPUF uses input from all the paths in the final output by using the frequency control bits as seen in Fig. 3. Thus, the attacker will gain some knowledge about the modified CaPUF internal status even when locking occurs. Hence, making the locking less successful as the attacker will collect some knowledge about the modified CaPUF.

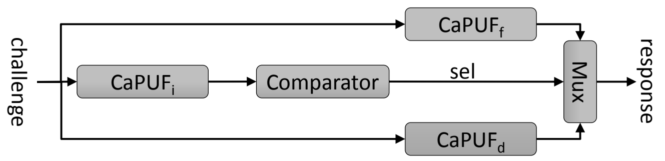


Fig. 6: Obfuscation Mode for CaPUF

CaPUF can also be used to perform obfuscation. Instead of disabling the output, the output comes from the extra PUF. Fig. 6 shows the implementation of CaPUF$_O$ that we use to implement the obfuscation by extending the design of CaPUF$_L$ with an additional dummy CaPUF (CaPUF$_d$) and a multiplexer. The output of the comparator controls the multiplexer. If the challenge is from the pre-authenticated challenges, the output comes from CaPUF$_f$. Otherwise, the output comes from CaPUF$_d$. Hence, an attacker will not be able to tell from which PUF the response comes and the collected CRPs will not help to build an accurate model, as they come from two different PUFs.

Both CaPUF$_f$ and CaPUF$_d$ have similar and high reliability. Hence, an attacker will not be able to get data based on the reliability difference. Moreover, the attacker will not be able to infer from the other responses from which CaPUF the output comes, as the challenge can get any random output from any of both CaPUFs. During the enrollment phase, it is important to know which responses come from CaPUF$_f$ and which responses come from CaPUF$_d$. Hence, the select signal from the comparator is used to perform the differentiation between both CaPUFs. Based on the select signal, the authenticated challenges will be identified as the ones getting a response by CaPUF$_f$. After the enrollment phase, this output has to be sealed as it would leak the obfuscation secret. For example, if the PUF is implemented on FPGA, then this output should not be connected to any pins or signals. Sealing extra outputs is not uncommon for PUFs. For example, LPN [16] requires to have access to the POK output known at the enrollment phase to fine-tune the implementation of the fuzzy extractor on chip. However, after the enrollment, the output of the POK must not be accessible outside the IC; only the hash function output is accessible. The same also applies for FSM, which needs the responses to fine-tune the finite state machine. More details about LPN and FSM are provided in Section III.

VI. Evaluation

A. Experimental Setup

After designing our CaPUF, we tested it in real hardware. First, CaPUF was implemented as synthesizable VHDL code. The target is the Xilinx VC707 board containing a Virtex-7 FPGA. Moreover, to ensure that the the PUF is not affected with modifications to the experimental setup framework, it was placed into a Pblock with constraints that manually place its elements to selected resources.

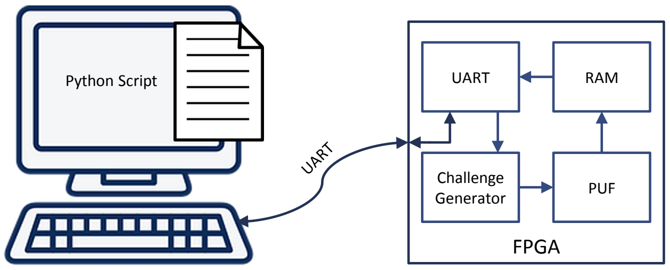


Fig. 7: The experimental setup for logging the challenge response pairs

We used four different VC707 boards to test our CaPUF and evaluate the different PUF metrics from Section II. Fig. 7 shows the whole framework for conducting the experiments. In addition to the PUF, we needed other components to be able to log the results and communicate them to a PC for further analysis. A challenge generator is used to send the challenges to the PUF and collect a large set of CRPs. The challenges should be random and not in sequence to cover as many possibilities as possible. We used an LFSR with a constant seed generated randomly as a PRNG to generate the challenges. The responses along the challenges are stored on the FPGA in block RAM. For each PUF implemented on one of the four boards, 65,536 CRPs are generated. Each CRP is generated 1000 times to evaluate the reliability. The CRPs are communicated via UART to the PC to perform the analysis.

In addition to the CaPUF, we implemented PLPUF [36], APUF, 3-XORPUF, and 5-XORPUF [17]. Both XORPUFs are based on the implemented APUF. The difference between them is the number of used PUFs for the xor (3 or 5). This comes with a trade-off between security and resource usage. We also recorded the performance of the PLPUF on all four boards. This is because we compare CaPUF to PLPUF, which is used as its building block to see if any of the metrics (uniqueness, reliability, and uniformity) degrades. APUF, 3-XORPUF, and 5-XORPUF were only evaluated on one of the boards, as we only use them to assess if the attacks work or not and this can be measured based on one board.

B. PUF Quality and Performance

After collecting the CRPs, we evaluated the quality and performance of CaPUF. This is an important step before trying to evaluate its performance against ML-modeling attacks. If it has a bad uniqueness, then an attacker can model it based on a reference PUF. If it has a biased uniformity, then an attacker can predict large portions of the response. If it has low reliability, then it is noise-dominated and cannot be used to authenticate a device.

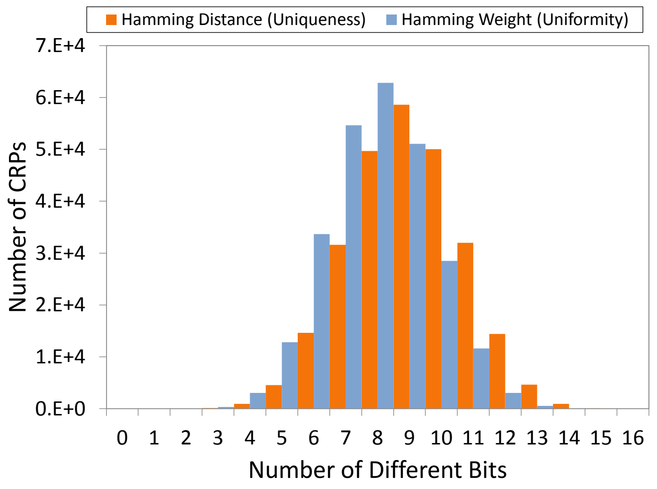


Fig. 8: Hamming weight representing uniformity and Hamming distance representing uniqueness based on all the values collected from the FPGA boards showing the desired normal distribution behavior of CaPUF

For uniqueness, Fig. 8 shows in orange the results for calculating the Hamming distance between the CaPUFs on the different boards. It has a Gaussian distribution which is to be expected for such a random relation. It has a peak at 9, which is only 1 bit away from the ideal 8. This is an acceptable deviation as it is still close to the 50% which would be desired. Overall calculating the average on this Hamming distance was 8.9 and the standard deviation was 1.76.

As for uniformity, Fig. 8 shows in blue the Hamming weight of the responses from the four boards. Here we also get the expected Gaussian distribution. In addition, the peak is exactly at 8 which is the ideal value. This means that for uniformity, CaPUF has an almost ideal performance. Overall it has 8.01 as an average and a standard deviation of 1.63.

TABLE I: Performance of PLPUF and CaPUF

PUF	Uniqueness	Uniformity	Reliability
PLPUF	49.60%	46.12%	98.47%
CaPUF	55.63%	50.06%	92.54%

Moreover, Table I shows the results of the reliability, uniformity, and uniqueness over all four boards based on the equations from Section II. The metrics are calculated both for PLPUF (the building block of CaPUF) and CaPUF. The uniqueness of PLPUF is slightly better than the uniqueness of CaPUF as it is closer to the 50% value, both are still under 60%. However, CaPUF has an improvement for the uniformity over PLPUF as it is almost the ideal 50% value. As for reliability, PLPUF was able to achieve a remarkably high value of 98.47%. CaPUF has a lower reliability, which can be understood as it is based on some chains of PUFs, which reduces the overall reliability. This does not affect the performance of CaPUF, as it is still in the range where fuzzy extractors can extract the response on the verifier side [2]. Note that CaPUF has the advantage of being ML-resilient over PLPUF, as we demonstrate next.

C. *Resilience against ML-based attack*

The collected CRPs are not only used to evaluate the PUF performance metrics but also to evaluate whether or not the ML-attacks would be successful against the PUF. We use the state-of-the-art open-source LR and SVM modeling attacks from [31] and train them based on the collected CRPs. The attack works as follows: It uses a separate model for each response bit with two classes representing both binary values. The challenge value is the single feature used for training. It performs 1000 iterations on the classifier's parameters (LR or SVM) to fine-tune the model.

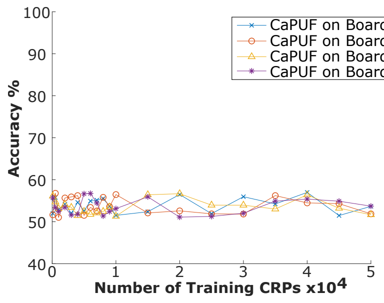
We use the attacks against all implemented PUFs. Up to 50,000 CRPs were used to train the models, the rest is used for testing and getting the prediction accuracy. We do not use a validation set, as we do not build the models ourselves but rather we use the already existing models from the state-of-the-art attacks. Hence, we did not perform any validation as we do not fine-tune the parameters but use the attack as it is. This is higher than the number of CRPs usually used for training, which is between 5,000 and 10,000 CRPs [31].

The results for both attacks are similar as shown in Fig. 9. APUF was easily breakable by both attacks. It required around 10,000 CRPs to be almost fully predictable and became partially predictable using even less CRPs. This is in line with many of the previous works [14], [15], [31]. PLPUF was also broken by both attacks. It showed to be harder to break than APUF, and it was easier to break it by using the LR attack. Note that to the best of our knowledge, our work is **the first work to report successful attacks** against PLPUF. We did not implement a new attack for this, but we used state-of-the-art attacks against PLPUF.

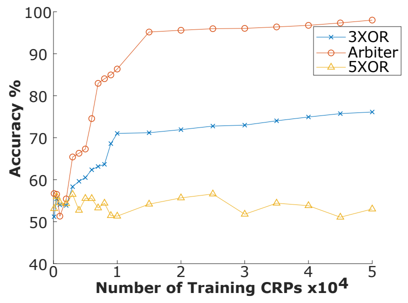
3-XORPUF was partially broken. With enough CRPs the bits were being predictable up to 75% accuracy. But neither of the attacks was able to fully model it. Only 5-XORPUF and our CaPUF were not broken by the attacks. The accuracy of the prediction did not reach higher than 57% for CaPUF and 59% for 5-XORPUF. Note that the prediction accuracy is per individual bit of the response i.e., the chance of correctly predicting every bit. This is why the ideal value is 50% (similar to flipping a coin on each bit). If the accuracy is too high or too low, then modeling the PUF worked well. For low accuracy, the attacker only needs to invert the values and then the bits will be of mostly correct values.

D. *Comparison to the State-of-the-Art*

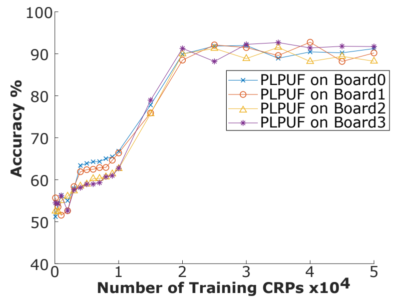
We extend our comparison to the other state-of-the-art works from Section III and we compare all three modes of our CaPUF with them. We do the comparison for all PUFs assuming they have 32 bit challenges and 16 bit responses, which are the same bit-widths as the ones from our CaPUF.



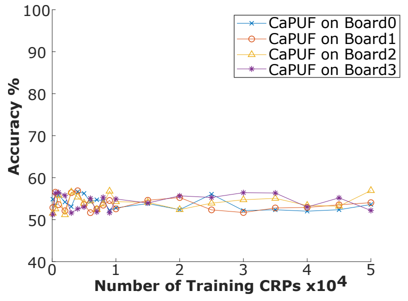
(a) LR on CaPUF



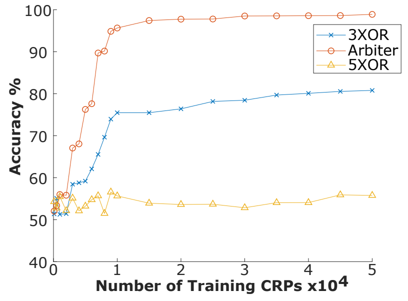
(b) LR on Arbiter-based PUFs



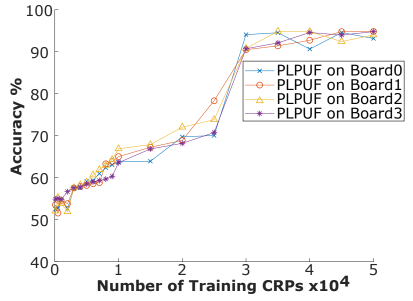
(c) LR on PLPUF



(d) SVM on CaPUF



(e) SVM on Arbiter-based PUFs



(f) SVM on PLPUF

Fig. 9: SVM and LR attacks on CaPUF, PLPUF, APUF, 3-XORPUF, and 5-XORPUF. The attacks are only as good as flipping a coin on CaPUF and 5-XORPUF, but they have a better performance against 3-XORPUF, and they are successful against APUF and PLPUF. Board0 till Board3 denote the four different boards used to implement the PUFs.

TABLE II: Comparison to the related works

PUF	ML-Resil.	Locking	LUTs	Memory	Time
CRC	yes	no	5120	0	N.A.
PLPUF	no	no	32	0	1 cycle
CaPUF	yes	no	329	0	16 cycles
CaPUF$_L$	yes	yes	637	0	32 cycles
CaPUF$_O$	yes	obfusc.	941	0	32 cycles
Arbiter	no	no	1024	0	1 cycle
3-XOR	partial	no	3072	0	1 cycle
5-XOR	yes	no	5120	0	1 cycle
Slender	yes	no	1168	68 KB	N.A.
CT	yes	no	3072	0	1 cycle
LPN	yes	no	49K	7 KB	N.A.
FSM	yes	yes	1082	68 KB	2508 cycles
IPA	yes	yes	1060	2 KB	18764 cycles
NoPUF	partial	obfusc.	1024	0	1 cycle

The numbers for area and latency are based on implementing them on FPGA, based on their description of their respective works or the numbers reported by the work itself. Hence, the numbers are in Lookup Tables (LUTs) and memory is in Kilobytes.

Our CaPUF is the smallest ML-resilient PUF. As a matter of fact, only PLPUF is smaller than CaPUF, even APUF requires more hardware than CaPUF. The closest in terms of area to our implementation from the ML-resilient PUFs is CT, which requires almost 10× more LUTs than our CaPUF. Slender requires fewer LUTs than CT, however, it uses 68 KBs of memory which would require more area.

It can be seen that all the cryptography-based PUFs have a significant resource usage. For example, the stream cipher dominates the resource usage of CRC. LPN has the highest resource usage, it is the only one that exceeds 10 K LUTs. This is because of its on-chip implementation of the error correction code. FSM and IPA both use significantly less hardware than LPN. However, both of them require thousands of clock cycles to complete their computation. The reason for this is that they use a lightweight serial hash function which is slow. Additionally for IPA, it uses voting rounds instead of error correction which adds a significant delay overhead. These delay overheads are very problematic in the enrollment phase, as collecting the CRPs requires three to four orders of magnitude longer time for each individual IC. The delay of our three CaPUF modes is also slightly higher than the other PUFs. However, it is similar to the serial implementations of weak PUFs when they try to output several bits, so it is in the acceptable ranges [38].

As for the locking and obfuscation modes, our CaPUF$_L$ has lower LUT usage than FSM and IPA Moreover, it is faster in terms of clock cycles and does not need any memory. For obfuscation, the resource usage of CaPUF$_O$ and NoPUF are very similar. However, as mentioned in Section III, NoPUF can be vulnerable to a knowledgeable attacker who uses reliability-based attacks to build its model. Our CaPUF$_O$ does not suffer from this, as the output comes from another PUF that has similar reliability to the original PUF, instead of using a noisy output as NoPUF does it. Improving NoPUF to implement obfuscation similar to ours would require at least a secure PUF for the first stage to avoid modeling it. This would result in a significant increase in resource usage (roughly 6× increase), which is avoided in our case as our CaPUF is lightweight. Note that we are not able to evaluate NoPUF experimentally. To implement NoPUF, one

TABLE III: CaPUF Sensitivity for the number of stages

Variant	Unique.	Unifrm.	Relib.	LUTs	Cycles	Predict.
3×4	51.20%	48.63%	96.72%	313	12	61.78%
$3\times4_{vc}$	51.66%	49.25%	99.10%	953	1536	63.34%
4×4	55.63%	50.06%	92.54%	329	16	57.15%
$4\times4_{vc}$	55.88%	49.81%	98.51%	969	2048	56.39%
5×4	50.47%	51.34%	85.90%	601	20	49.06%
$5\times4_{vc}$	49.86%	52.10%	97.63%	1883	2560	47.99%
6×4	49.21%	53.18%	81.67%	1127	24	51.88%
$6\times4_{vc}$	49.34%	53.06%	97.04%	3705	3072	52.46%

needs to have access to the fabrication phase of the chip, as the NoPUF idea is based on fine-tuning some parameters during the fabrication process. Without this fine-tuning, we are only able to implement a noisy PUF without reliable CRPs, which would lead to an unfair comparison.

E. Sensitivity to Number of Stages and Generalization

We chose to implement our CaPUF having four stages. However, based on our analysis in Section IV-C it should be possible to implement it using three stages or more. We analyze the sensitivity of CaPUF to the number of stages. Table III shows the results of this analysis. We first reduce the number of stages from 4 to 3. The PUF is more predictable by the attacks with the removal of this stage. The difference in hardware is minimal and does not show a real advantage. We then try to add a fifth and a sixth stage consecutively. We use voting cycles to enhance the reliability as without the voting cycles the reliability is highly affected. Voting cycles are achieved by adding simple counters and performing the same operation 127 times. Then, based on the counter value, the value of the PUF response is recorded. If the output value is 1 for 64 or more times, then the bit is collected as 1, otherwise, it is collected as 0. We tried this voting cycles options for all four variants and name them in Table III by giving them the subscript vc. The reliability improves significantly, however, this comes with a linear overhead in delay and resources. Therefore, the 4×4 CaPUF has the best trade-off between reliability, predictability, and overhead.

Moreover, in Section IV-D we mention that the concept of CaPUF can be extended to other PUFs and give an example of such generalization using APUF under the name CaPUF_{AP}. To evaluate this, we implement CaPUF_{AP} and evaluate its performance. It has the same challenge and response widths of the normal CaPUF based on PLPUF. Table IV shows the results of this implementation. In general, the performance of CaPUF_{AP} is slightly worse compared to CaPUF. However, the main disadvantage is the resource usage. CaPUF_{AP} has a significantly higher overhead around 17×. This is normal and was expected as the APUF already has a significantly higher resource usage compared to the PLPUF.

TABLE IV: CaPUF behavior when implemented based on APUF instead of PLPUF

Uniqueness	Uniformity	Reliability	LUTs	Prediction
47.03%	46.38%	88.47%	5723	61%

VII. Conclusion

In this work, we tackle the problem of ML-based attacks against PUFs. We proposed and presented our novel CaPUF. CaPUF achieves ML resiliency by cascading the inputs and outputs of several lightweight PUFs, as well as by controlling the frequency of each lightweight PUF based on the output from other PUFs. We show a mathematical model to explain why it is hard to model CaPUF through ML attacks. Moreover, we run state-of-the-art attacks against CaPUF and show that CaPUF is resilient against them. Using the same attacks we are the first to report that PLPUF is breakable by ML attacks. Additionally, in comparison to the existing ML-resilient PUFs, CaPUF has the smallest area and can operate by locking or obfuscating the output as an extra layer of security.

Acknowledgment

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Invasive Computing" (SFB/TR 89).

We also acknowledge Zhiyao Wang whose work on implementing PLPUF was used as a base for building our novel CaPUF.

References

[1] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1074376

[2] J. Kong, F. Koushanfar, P. K. Pendyala, A.-R. Sadeghi, and C. Wachsmann, "Pufatt: Embedded platform attestation based on novel processor-based pufs," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–6.

[3] L. Bolotnyy and G. Robins, "Physically unclonable function -based security and privacy in RFID systems," *Proceedings - Fifth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2007*, no. September, pp. 211–218, 2007.

[4] N. A. Anagnostopoulos, T. Arul, Y. Fan, C. Hatzfeld, J. Lotichius, R. Sharma, F. Fernandes, F. Tehranipoor, and S. Katzenbeisser, "Securing iot devices using robust dram pufs," in *2018 Global Information Infrastructure and Networking Symposium (GIIS)*, 2018, pp. 1–5.

[5] J. Calhoun, C. Minwalla, C. Helmich, F. Saqib, W. Che, and J. Plusquellic, "Physical unclonable function (puf)-based e-cash transaction protocol (puf-cash)," *Cryptography*, vol. 3, no. 3, 2019. [Online]. Available: https://www.mdpi.com/2410-387X/3/3/18

[6] E. Abtioglu, R. Yeniceri, B. Govem, E. Goncu, M. E. Yalcin, and G. Saldamli, "Partially reconfigurable IP protection system with ring oscillator based physically unclonable functions," *Proceedings - 2017 1st New Generation of CAS, NGCAS 2017*, pp. 65–68, 2017.

[7] S. Kleber, F. Unterstein, M. Matousek, F. Kargl, F. Slomka, and M. Hiller, "Design of the Secure Execution PUF-based Processor (SEPP)," *Workshop on Trustworthy Manufacturing and Utilization of Secure Devices, TRUDEVICE 2015*, no. 2, pp. 1–5, 2015.

[8] S. Kleber, F. Unterstein, M. Hiller, F. Slomka, M. Matousek, F. Kargl, and C. Bösch, "Secure Code Execution: A Generic PUF-Driven System Architecture," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11060 LNCS, pp. 25–46, 2018.

[9] R. Nithyanand and J. Solis, "A theoretical analysis: Physical unclonable functions and the software protection problem," *Proceedings - IEEE CS Security and Privacy Workshops, SPW 2012*, no. May 2012, pp. 1–11, 2012.

[10] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure pufs," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 670–673.

[11] M. A. Qureshi and A. Munir, "PUF-IPA: A PUF-based Identity Preserving Protocol for Internet of Things Authentication," *2020 IEEE 17th Annual Consumer Communications and Networking Conference, CCNC 2020*, 2020.

[12] E. Dubrova, O. Naslund, B. Degen, A. Gawell, and Y. Yu, “CRC-PUF: A Machine Learning Attack Resistant Lightweight PUF Construction,” *Proceedings - 4th IEEE European Symposium on Security and Privacy Workshops, EUROS and PW 2019*, pp. 264–271, 2019.

[13] C. Yehoshuva, R. R. Adhithan, and N. N. Anandakumar, “A survey of security attacks on silicon based weak puf architectures,” in *SSCC*, 2020.

[14] G. Hospodar, R. Maes, and I. Verbauwhede, “Machine learning attacks on 65nm Arbiter PUFs: Accurate modeling poses strict bounds on usability,” *WIFS 2012 - Proceedings of the 2012 IEEE International Workshop on Information Forensics and Security*, pp. 37–42, 2012.

[15] G. T. Becker, “On the Pitfalls of Using Arbiter-PUFs as Building Blocks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1295–1307, 2015.

[16] C. Jin, C. Herder, L. Ren, P. Nguyen, B. Fuller, S. Devadas, and M. van Dijk, “FPGA Implementation of a Cryptographically-Secure PUF Based on Learning Parity with Noise,” *Cryptography*, vol. 1, no. 3, p. 23, 2017.

[17] G. E. Suh and S. Devadas, “Physical unclonable functions for device authentication and secret key generation,” in *2007 44th ACM/IEEE Design Automation Conference*, 2007, pp. 9–14.

[18] A. Wang, W. Tan, Y. Wen, and Y. Lao, “NoPUF: A Novel PUF Design Framework Toward Modeling Attack Resistant PUFs,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 6, pp. 2508–2521, 2021.

[19] Y. Gao, H. Ma, S. F. Al-Sarawi, D. Abbott, and D. C. Ranasinghe, “PUF-FSM: A Controlled Strong PUF,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 5, pp. 1104–1108, 2018.

[20] X. Lu, L. Hong, and K. Sengupta, “Cmos optical pufs using noise-immune process-sensitive photonic crystals incorporating passive variations for robustness,” *IEEE Journal of Solid-State Circuits*, vol. 53, no. 9, pp. 2709–2721, 2018.

[21] Y. Lin, Y. Zhang, J. Chen, and J. Han, “A novel lightweight pufs using interconnect line mismatch for hardware security,” in *2021 China Semiconductor Technology International Conference (CSTIC)*, 2021, pp. 1–2.

[22] H. R. Ghaeini, M. Chan, R. Bahmani, F. Brasser, L. Garcia, J. Zhou, A.-R. Sadeghi, N. O. Tippenhauer, and S. Zonouz, “PAtt: Physics-based attestation of control systems,” in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. Chaoyang District, Beijing: USENIX Association, Sep. 2019, pp. 165–180. [Online]. Available: https://www.usenix.org/conference/raid2019/presentation/ghaeini

[23] O. Willers, “Mems sensors as physical unclonable functions,” 2019.

[24] K. Phalak, A. A. Saki, M. Alam, R. O. Topaloglu, and S. Ghosh, “Quantum puf for security and trust in quantum computing,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 333–342, 2021.

[25] S. Gehrer and G. Sigl, “Using the reconfigurability of modern FPGAs for highly efficient PUF-based key generation,” *10th International Symposium on Reconfigurable and Communication-centric Systems-on-Chip, ReCoSoC 2015*, 2015.

[26] J. H. Anderson, “A PUF design for secure FPGA-based embedded systems,” *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pp. 1–6, 2010.

[27] Y. Komano, K. Ohta, K. Sakiyama, M. Iwamoto, I. Verbauwhede, and D. Schneider, “Single-round pattern matching key generation using physically unclonable function,” *Security and Communication Networks*, vol. 2019, no. Id, 2019.

[28] A. Schaller, W. Xiong, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser, and J. Szefer, “Intrinsic Rowhammer PUFs: Leveraging the Rowhammer effect for improved security,” *Proceedings of the 2017 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2017*, pp. 1–7, 2017.

[29] R. Canetti, B. Fuller, O. Paneth, L. Reyzin, and A. Smith, “Reusable Fuzzy Extractors for Low-Entropy Distributions,” *Journal of Cryptology*, vol. 34, no. 1, pp. 1–33, 2021. [Online]. Available: https://doi.org/10.1007/s00145-020-09367-8

[30] A. Herkle, P. Rossak, H. Mandry, J. Becker, and M. Ortmanns, “Comparison of measurement and readout strategies for ro-pufs on xilinx zynq-7000 soc fpgas,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[31] Q. Wu and J. Zhang, “CT PUF: Configurable tristate PUF against machine learning attacks,” *Proceedings - IEEE International Symposium on Circuits and Systems*, vol. 2020-Octob, 2020.

[32] F.-J. Streit, P. Krüger, A. Becher, J. Schlumberger, S. Wildermann, and J. Teich, “Choice – a tunable puf-design for fpgas,” in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 38–44.

[33] P. H. Nguyen, D. P. Sahoo, C. Jin, K. Mahmood, and M. van Dijk, “Mxpuf: Secure puf design against state-of-the-art modeling attacks,” *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 572, 2017.

[34] Y. Tang, D. Wu, Y. Cao, and M. Margraf, “The Shift PUF: Technique for Squaring the Machine Learning Complexity of Arbiter-based PUFs: Work-in-Progress,” *Proceedings of the 2020 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES 2020*, pp. 9–11, 2020.

[35] T. Kroeger, W. Cheng, S. Guilley, J. L. Danger, and N. Karimi, “Cross-PUF Attacks on Arbiter-PUFs through their Power Side-Channel,” *Proceedings - International Test Conference*, vol. 2020-Novem, pp. 1–5, 2020.

[36] Y. Hori, H. Kang, T. Katashita, and A. Satoh, “Pseudo-LFSR PUF: A compact, efficient and reliable physical unclonable function,” *Proceedings - 2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011*, pp. 223–228, 2011.

[37] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, “Slender PUF protocol: A lightweight, robust, and secure authentication by substring matching,” *Proceedings - IEEE CS Security and Privacy Workshops, SPW 2012*, no. August 2014, pp. 33–44, 2012.

[38] N. N. Anandakumar, M. S. Hashmi, and S. K. Sanadhya, “Efficient and Lightweight FPGA-based Hybrid PUFs with Improved Performance,” *Microprocessors and Microsystems*, vol. 77, p. 103180, 2020. [Online]. Available: https://doi.org/10.1016/j.micpro.2020.103180