Virtual PUF: Built-in Model for Highly Reliable and Secure PUF

Neelam Nasir[1], Wei Cheng[2,1], Ulrich Kühne[1]
Tarik Graba[1], and Jean-Luc Danger[1]

[1] LTCI, Télécom Paris, Institut Polytechnique de Paris, 91120, Palaiseau, France
name.surname@telecom-paris.fr, wei.cheng@njust.edu.cn
[2] CSE, Nanjing University of Science and Technology, 210094, Nanjing, China

Abstract. Strong Physical Unclonable Functions (PUFs) with challenge-response protocols provide a cost-effective authentication solution for resource-limited devices. However, they are susceptible to modeling attacks. An effective countermeasure for multi-bin PUFs – such as the Ring Oscillator PUF (RO-PUF) and Loop PUF – is the use of Non-Monotonic Quantization (NMQ) of the response. However, NMQ requires quite high quantization levels in order to effectively improve the security. This makes the PUF unreliable, rendering it impractical for authentication purposes. In this paper, we present a solution called the *Virtual PUF*: It generates a lightweight PUF model at device start-up, which is then used instead of the physical PUF during authentication. This allows the elimination of the noise impact and renders the PUF fully reliable and secure against ML attacks at the same time. As a proof of concept, we present an FPGA implementation based on a Loop PUF – a PUF relying on a single ring oscillator – and we show that beside its perfect reliability and high security, the Virtual PUF can be designed in a lightweight manner. However, the model built by the Virtual PUF may differ from the enrolled model and cause mismatch errors between the two models, which could deteriorate the authentication protocol. We show that the mismatch error measured in different configurations and environments can be optimized towards an acceptable range and/or managed by the authentication protocol.

Keywords: Hardware Security · Physical Unclonable Functions · Machine-learning Attacks · Reliability · Non-Monotonic Quantization · Virtual PUF · Authentication

1 Introduction

Strong Physical Unclonable Functions (PUFs) have been proposed as a low-cost security anchor [9, 17], for lightweight authentication by using Challenge-Response Pairs (CRP) without additional cryptographic functions to meet low-cost requirements. In the context of CRP authentication, for a PUF to be *secure*, its output must be hard to predict. Indeed, there are attacks relying on machine learning (ML) techniques to model the PUF from a set of recorded CRPs [13].

In particular, PUFs relying on *delay chains* as entropy source – such as the Arbiter PUF [5] – have an inherently linear behavior and are therefore easily attackable by linear regression or more advanced ML techniques. In order to prevent such attacks, it is necessary to introduce non-linearity into the design. As an example, the XOR-PUF [17] takes advantage of a composition of Arbiter PUFs whose outputs are XORed. The Interpose-PUF [11] is another composed and robust strong PUF. Nevertheless, it is still attackable [19] with a divide and conquer strategy at the cost of an increased number of training challenges.

An alternative approach, which can be applied to PUFs having a multi-bin response, is called *Non-Monotonic Quantization* (NMQ) [16]. Multi-bin PUFs are able to output a response with n-bit entropy with a linear quantization. They are also called Alphabet PUFs [6] and can be implemented e.g. based on the Ring-oscillator PUF (RO-PUF) [17] or Loop PUF [2]. Using NMQ, the output space of the multi-bin PUF is partitioned into multiple zones, which are mapped to 1 and 0 in an interleaved fashion, thus generating only one bit output in a non-monotonic way. This makes the output value much harder to predict.

However, while the resistance against ML grows with the number of zones – denoted by the parameter Q – the reliability decreases rapidly. This is due to measurement noise that can lead to unstable responses, especially when the measured value is close to the border of one of the quantization intervals. If used in a challenge-response protocol, this poses a problem: In order to authenticate a device, more CRPs are needed to account for frequent errors in the responses. This will, in turn, expose more information to a potential attacker, who can incorporate it into their training data. Even worse, recent ML attacks have succeeded in leveraging reliability information to model a PUF with fewer CRPs [18].

As a remedy, we propose a new PUF architecture relying on NMQ quantization, which aims at perfect reliability and more robustness against ML attacks. The basic idea is to construct a model of the delay chain internally, at start-up. This model is then used instead of the physical PUF to produce responses in a deterministic way. This approach, which we call *Virtual PUF*, allows us to achieve higher resistance against ML attacks with higher Q values.

The design of the Virtual PUF poses a few challenges: Since the model must not be accessible outside the PUF to avoid compromising its security, it has to be constructed in hardware. This involves some rather complex operations such as matrix inversion and computing the standard deviation of a Gaussian distribution. In this paper, we propose several optimizations and approximation techniques that allow us to implement a Virtual PUF with low complexity overhead. The effectiveness of the approach has been validated using a Loop PUF implemented in an FPGA. A potential limitation is the *mismatch* between the model built internally by the Virtual PUF and the one enrolled on the server side. Indeed, since the model is reconstructed at each start-up of the device, changing environmental conditions – combined with artifacts from approximation and rounding – can lead to a slightly different model. This mismatch error has to be analyzed to validate the feasibility of the authentication protocol.

As a summary, the contributions of this paper are the following:

1. We propose a fully reliable PUF, the *Virtual PUF*, by replacing the physical model with an internal mathematical model.
2. we analyze ML attacks against the Virtual PUF and show its resistance at high Q.
3. we propose a lightweight implementation which has been designed on an FPGA for validation.
4. we validate the concept for different configurations and assess the mismatch error to adapt the authentication protocol.

The remainder of this paper is structured as follows. We will start with a discussion of related work in Section 2. Section 3 introduces the basic principles and properties of the Virtual PUF, in particular regarding its security and reliability. The overall architecture and hardware design are presented in Section 4. We show detailed evaluation results of the implementation in Section 5 before concluding the paper in Section 6 .

2 Related Work

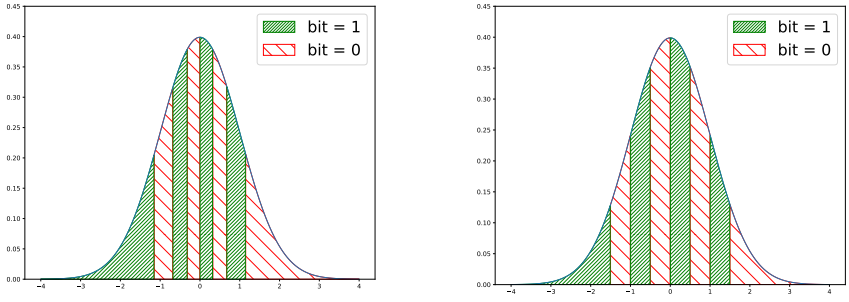
In this section, we present the context related to the ML attacks against PUF when used for authentication with CRPs. The protections against such attacks are either to build a natively robust PUF or to devise specific PUF protocols. The challenge is all the greater if the PUF has to keep its lightweight property.

2.1 Modeling Attacks

The modeling attack, or ML attack, allows the adversary to build a model of the PUF by using powerful ML algorithms and a minimum number of challenge-response pairs to learn. This allows the adversary to impersonate the PUF during authentication. In the literature dedicated to ML attacks against PUF CRP protocols, the challenges and responses are generally considered public. This corresponds to a very strong attacker model as the adversary can not only spy the entire protocol but can also reproduce it to exploit the reliability of the response. The Arbiter PUF [7] is one of the first silicon PUFs that has been devised and attacked by the Support Vector Machine (SVM) algorithm [8]. More derivatives of Arbiter PUF have been attacked using Logistic Regression (LR) in [13]. These attacks take advantage of the quasi-linear behavior of the Arbiter PUF, which relies on a delay chain.

2.2 Robust PUFs against ML Attacks by Composition

Increasing the non-linearity of the PUF behavior leverages the resistance against modeling attacks. A first method to achieve this property is to combine multiple PUFs. The XOR-arbiter PUF is a well-known composed PUF [17]. It relies on a



(a) Equiprobable Quantization (b) Equidistant Quantization

Fig. 1: Non-monotonic quantization with $Q = 8$.

set of Arbiter PUFs whose responses are XORed. The robustness is greatly increased as the number of required CRPs increases exponentially with the number of XORed PUFs. However, the exploitation of the response reliability by ML, e.g. using a Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [1] reduces in a linear manner the need for CRPs to learn. The Interpose PUF [11] is one of the most robust PUFs. But using a divide-and-conquer approach [19], and/or using the reliability information [18], greatly enhances the efficiency of ML attacks.

2.3 Robust PUFs against ML Attacks by Quantization

Another method to get a non-linear behavior is to quantize the raw PUF response in a non-monotonic way as proposed in [16]. This is possible with multi-bin PUFs such as the RO-PUF or the Loop PUF, where the raw output is an integer value that can be quantized with respect to Q quantization intervals. The final 1-bit response is obtained by interleaving the '0' and '1' intervals as shown in Figure 1 for $Q = 8$ where the intervals are equiprobable quantiles of the distribution in Figure 1a or equidistant in Figure 1b.

It has been observed in [10, 16] that this method improves the robustness against ML attacks for a sufficiently high value of Q. However, this makes the PUF highly unreliable. This can be explained by the increasing number of quantization thresholds, which increases the probability that environmental noise makes the raw response cross a threshold to an adjacent interval and thereby flips the response value. The study in [10] proposes a protocol that uses reliability information on the server side to mitigate this weakness.

3 Virtual PUF Principle

This section explains the main lines of the Virtual PUF with its potential advantages and limitations.

3.1 Concept

Built-in Model NMQ provides good security against ML attacks when Q is high, but is hardly usable because of noise. The proposed concept relies on using a built-in model rather than the real PUF during authentication, thus avoiding the unreliability caused by noise. The Virtual PUF model is built at every power-up by measuring the real physical PUF. The advantage lies in the elimination of noise, which allows to have a perfect reliability and to increase security by using a high quantization level Q. Furthermore, the inference time is greatly reduced compared to the long measurements needed by RO-based PUFs. However, building a model adds complexity and latency at start-up. Our study shows it is possible to create a model with simple hardware, thus allowing us to keep the lightweight property of the PUF. It should be noted that we consider only ML attacks in this paper. Building a PUF model at boot time may introduce side-channels, which are not considered in this study.

Enrollment Phase The enrollment of the PUF, once fabricated, is the same as that of an ordinary physical PUF. A trusted server registers the PUF model as well as the quantization thresholds for Q NMQ intervals.

Operational Phase At power-up, the internal model is created by using the physical PUF. The delays composing the delay chain and the thresholds for NMQ quantization are the main parameters of the model. Once computed, they are stored in memory to be read at every PUF operation. No data is stored in accessible or non-volatile memory. The detailed architecture is presented in section 4. It shows that the complexity of building the model can remain low by taking advantage of Hadamard challenges and other architectural considerations.

Authentication Protocol The authentication can be impacted by the mismatch between the enrolled model and the Virtual PUF model, notably because of environmental changes. We study the impact of these discrepancies in Section 5 and show that it can be managed by the protocol.

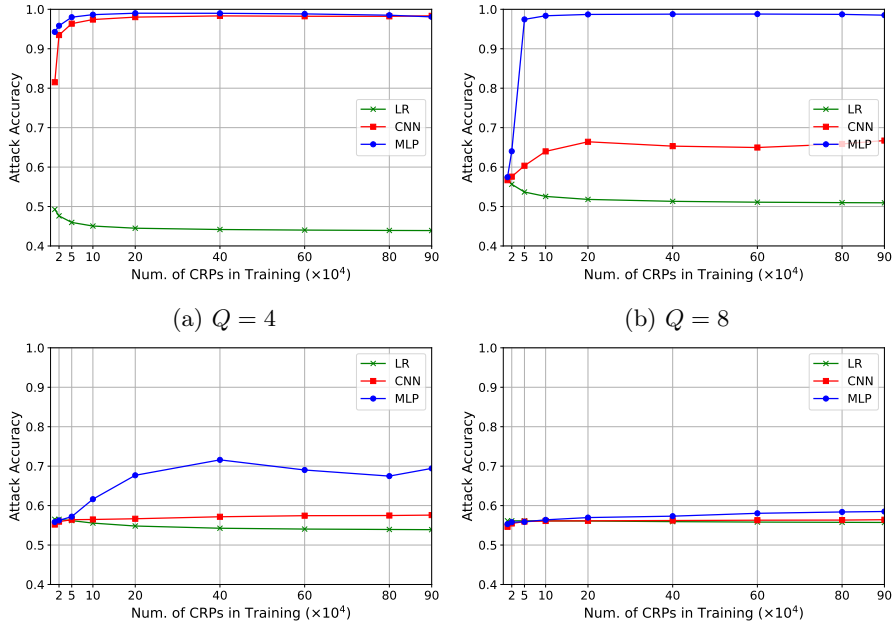
3.2 Security against Modeling Attacks

An important assumption that justifies the Virtual PUF concept is that NMQ with high values of Q provides a significant enhancement in security against ML attacks. We verify this assumption by applying modeling attacks based on Neural Networks (NNs), including Convolutional Neural Networks (CNNs), and the Multi-Layer Perceptron (MLP) as attacking means. In addition, we use Logistic regression (LR) as the baseline for comparison.

Regarding the considered attack model, the adversary is assumed to have access to any set of challenge-response pairs (CRPs) – often measurable CRPs under normal operating conditions – but not to internal physical parameters or power-on/off transient states. The attacker uses this input-output data to train

a mathematical or ML model that approximates the PUF's behavior, enabling the prediction of responses to unknown challenges.

The results of the modeling attacks are shown in Figure 2 for various Q values from 4 to 32 by using at most $900,000$ CRPs for model training, and a non-overlapping set of $100,000$ CRPs for testing the attack. It should be noted that all accuracy results are obtained by averaging on 30 repeated experiments. As shown in Figures 2c and 2d, with $Q \geq 16$, the Virtual PUF shows excellent resistance against modeling attacks, even enhanced with neural networks. These results validate our assumption, motivating our Virtual PUF design. Note that for these experiments, we used equal distances (as shown in Figure 1b) for the quantiles rather than equal probabilities (as shown in Figure 1a), which corresponds to the technique used in our Virtual PUFs as described in detail in Section 4.4.

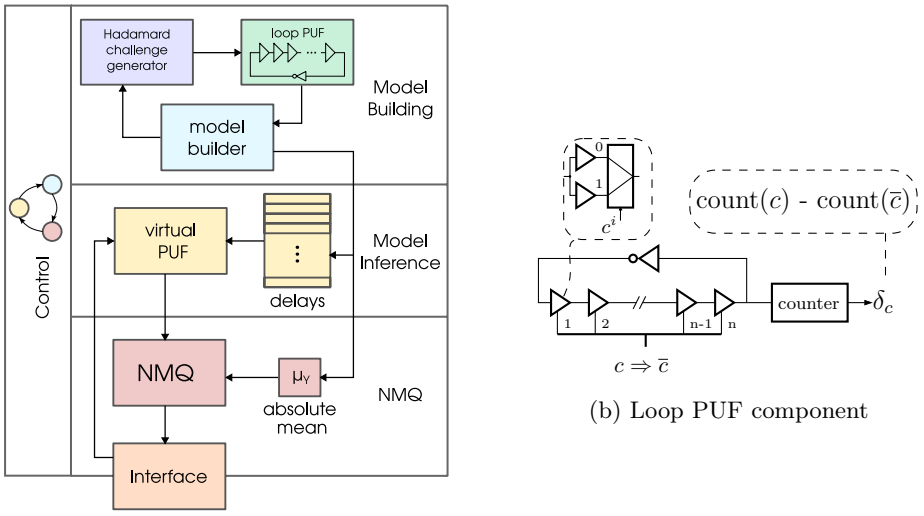


(a) $Q = 4$ (b) $Q = 8$

(c) $Q = 16$ (d) $Q = 32$

Fig. 2: Experimental results of modeling attacks with LR, CNN and MLP against Virtual PUFs with different Q.

It should be noted that the proposed solution is inherently resistant to modeling attacks that use reliability information to enhance the success rate as CMA-ES [1]. Once established at power-up, the Virtual PUF is fully deterministic and does not give away any reliability information on individual challenges. We could also consider ML attacks exploiting reliability information by rebooting the PUF multiple times. However, this hypothesis has not been taken into account in this



(a) Virtual PUF architecture

(b) Loop PUF component

Fig. 3: Hardware architecture of the Virtual PUF.

paper for different reasons: first, the physical access needed to reboot the PUF may be limited or impossible; second, the boot time may be prohibitive if many CRPs are necessary; finally, no current study of reliability attacks on NMQ has been published to assess the number of required learning CRPs. We will consider such attacks in future work.

4 Virtual PUF Architecture

In this section, we present the overall architecture of the proposed PUF design, followed by a detailed discussion on the different modules that constitute the Virtual PUF.

4.1 Overview of the Architecture

Figure 3a shows a block diagram of the Virtual PUF design. At its center is the model builder, which will populate the internal PUF model at power-up. The physical PUF is a Loop PUF [2], which is a simple and easy-to-design multi-bin PUF. The model builder uses a set of standardized challenges – Hadamard challenges – in order to query the Loop PUF, and derive the delay values and additional parameters for the quantization. Any subsequent user challenge is treated by the Virtual PUF module, which uses the internal model to calculate the response and outputs the quantized response to the user.

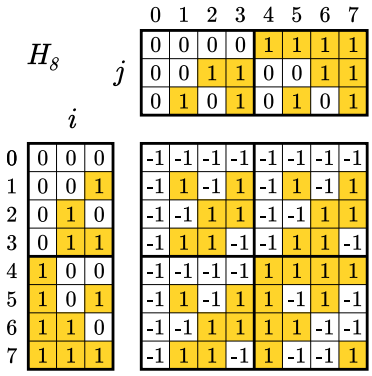


Fig. 4: Construction of the H_8 Hadamard matrix.

4.2 Virtual PUF Model Building

Loop PUF The Loop PUF is a multi-bin PUF [2] relying on a single ring oscillator composed of n controlled delay elements as shown in Figure 3b. Every delay element is controlled by a challenge bit, selecting one of two possible paths. In this way, the overall delay of the complete path is determined by the challenge vector $c \in \{0,1\}^n$. The PUF operation consists of counting the number of oscillations during a constant time window w with firstly the challenge c and secondly the complementary challenge $\overline{c}$. Computing the *difference* between the two counting results gives the raw integer response called δ_c. The response δ_c as well as the delays are normally distributed [5] as they represent a difference between the delay for c minus the delay for $\overline{c}$.

Model Builder The raw response δ_c of a Loop PUF to a challenge c can be expressed by a vector product:

$$\delta_c = \hat{c} \cdot \hat{D} \tag{1}$$

Here, we denote by $\hat{c}$ the challenge vector c with all zeros replaced by -1, and $\hat{D} \in \mathbb{R}^n$ is a vector representing the n delay differences of the Loop PUF. The model of the vector $\hat{D}$ is approximated into a fixed-point vector D, $D \in \mathbb{Z}^n$.

If the number of delays n is chosen as a power of two ($n = 2^k$), the delay values composing the chain of the Loop PUF can be computed by using a set of n challenges of n bits forming an Hadamard matrix $H \in \mathbb{Z}^{n \times n}$ composed of ± 1 values. The n Hadamard challenges form an orthogonal basis which enables us to characterize the entropy of the Loop PUF as demonstrated in [12, 15] and exploited to generate a cryptographic key [3, 14].

As the Hadamard matrix H is invertible, if we apply n challenges corresponding to the rows of H, we obtain the delay profile D through matrix inversion:

$$D = H^{-1} \Delta_H \tag{2}$$

where $\Delta_H = (\delta_{H_0}, \delta_{H_1}, \ldots, \delta_{H_{n-1}})$ is the response vector[3] for the n Hadamard challenges, given by the rows of the matrix H. However, direct matrix inversion can be computationally prohibitive for hardware implementation. This limitation can be overcome by exploiting a property of the Hadamard matrices, where the inverse can be computed simply through matrix transposition and scaling:

$$H^{-1} = \frac{1}{n} H^T \quad (\text{since } HH^T = nI) \tag{3}$$

The delay calculation can be further optimized for hardware implementation through three key transformations. First, the matrix transpose operation can be eliminated because $H^T = H$ for symmetric Hadamard matrices. Second, we exploit the binary nature of Hadamard challenges (± 1) to replace multiplications with conditional additions:

$$D_i = \frac{1}{n} \sum_{j=1}^{n} \begin{cases} +\delta_{H_j}, & \text{if } H_{ij} = +1 \\ -\delta_{H_j}, & \text{if } H_{ij} = -1 \end{cases} \tag{4}$$

Finally, as $n = 2^k$, the division operation can be realized at virtually no cost in hardware. We can either choose to drop the lower k bits or keep $\ell \leq k$ bits as the fractional part of a fixed-point representation. This optimized implementation requires only simple arithmetic operations, making it ideal for resource-constrained hardware implementations while maintaining mathematical equivalence to the original formulation.

Hadamard Challenge Generator The generation of the n vectors of the Hadamard matrix can be computed by Algorithm 1 by means of the binary scalar product of two index variables.

Algorithm 1 Hadamard matrix generation

FUNCTION generates Hadamard matrix H_n, $n = 2^k$
for $i \leftarrow 0$ **to** $2^k - 1$ **do**
  for $j \leftarrow 0$ **to** $2^k - 1$ **do**
    $\boldsymbol{i} \leftarrow \text{bin}(i)$ {$\boldsymbol{i}$ is a k-bit vector}
    $\boldsymbol{j} \leftarrow \text{bin}(j)$ {$\boldsymbol{j}$ is a k-bit vector}
    $p \leftarrow \bigoplus_{\ell=0}^{k-1} \boldsymbol{i}_\ell \cdot \boldsymbol{j}_\ell$
    $H[i][j] \leftarrow \begin{cases} 1, & \text{if } p = 1 \\ -1, & \text{if } p = 0 \end{cases}$ {(0,1) ⇒ (-1,+1)}
  end for
end for
return H_n

[3] Note that in practice, it is necessary to perform multiple measurements and use the *average responses* in order to eliminate noise.

The scalar product (line 6 in Algorithm 1) uses the binary representation of the indices i and j (which is the actual representation in hardware). The product can be computed as the exclusive OR of the conjunction (Boolean AND) of the two bit vectors, which can be realized with very few logic gates in a single clock cycle. The product is mapped to ± 1 in order to obtain the value of H_{ij}. As an illustration for this Algorithm, Figure 4 shows the construction of H_8.

4.3 PUF Model Inference

Having computed and stored the delay vector D, we can now infer a response to a given challenge without running the actual loop PUF, by simply computing Equation (1) by using D instead of the physical delay vector $\hat{D}$. Since $\hat{c}$ only contains values ± 1, we can again use conditional addition and avoid more costly integer multiplication.

4.4 Non-Monotonic Quantization

NMQ is an important building block in the PUF design in order to improve its resistance against ML attacks. Given the raw response δ_c and – even – quantization level Q, the Non-Monotonic Quantization can be defined as:

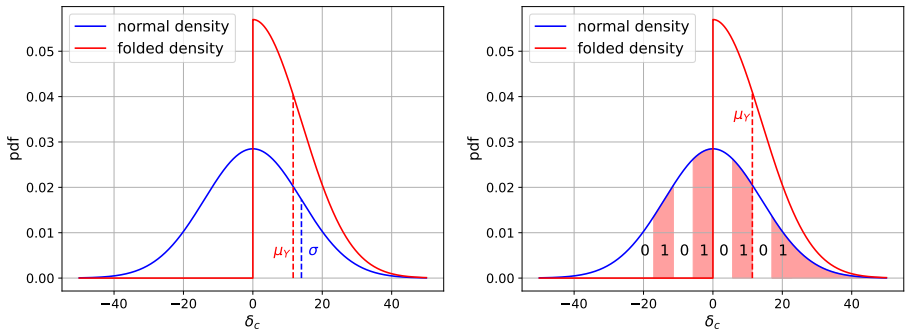
$$\mathrm{NMQ}_Q(\delta_c) = \begin{cases} 1, & \text{if } T_{2i-1} < \delta_c \leq T_{2i}, \text{ for some } i \in \{1, \ldots, \frac{Q}{2}\} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where the T_i are threshold values delimiting the intervals of the distribution.

Usually, *equiprobable* quantiles are used in order to obtain an unbiased response. This requires estimating the standard deviation of the raw response's distribution in order to compute the threshold values. Unfortunately, this is a costly operation when implemented in pure hardware. Note, however, that for an unbiased response, the quantiles need not necessarily be of equal probability. As an alternative, evenly spaced thresholds can be used, centered around the mean value μ. While this gives higher weight to the quantiles near the center of the distribution, it simplifies the calculation of the threshold values, allowing for an efficient hardware implementation. The threshold values still need to be fitted to the range of the PUF responses.

We propose a hardware-efficient solution to calculate the thresholds that uses the *absolute mean* of the *folded distribution* of the raw PUF responses. The *folded normal distribution* describes the absolute value $Y = |X|$ of a normally distributed random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ as shown in Fig. 5a. As can be seen in the figure, the mean of the folded distribution μ_Y is reasonably close to the standard deviation σ of the original distribution. We will therefore use it as a means to adapt the threshold values to the range of the raw PUF responses.

The following are the steps to implement hardware-friendly NMQ on an FPGA:



(a) Folded normal distribution (b) Thresholds for $Q = 8$ derived from μ_Y

Fig. 5: Raw response distribution and thresholds.

Calculation of folded mean μ_Y In order to estimate the folded mean, we use the n Hadamard challenges:

$$\mu_Y = \frac{1}{n}\left(\sum_{i=0}^{n-1} |\delta_{H_i}|\right)$$

Here, n (the size of the loop PUF) is chosen as a power of two (in our case with $n = 64$). The division by n can therefore be performed by right-shifting the sum by $\log_2(n)$ bits, which is a zero-cost operation in hardware (see Algorithm 2). For efficiency, the computation of the absolute mean is merged with the delay computations (see Section 4.2), since both are based on the collected responses Δ_H to the Hadamard challenges.

Algorithm 2 Absolute Mean of PUF folded mean distribution

```
FUNCTION compute_mean_abs (Hadamard_responses)
abs_sum ← 0
for i ← 0 to n − 1 do
    abs_sum ← abs_sum + |δ_{H_i}|
end for
μ_Y ← abs_sum ≫ log_2(n)
return μ_Y
```

Threshold calculation Given the absolute mean μ_Y, we can calculate the thresholds that divide the raw output range into Q quantiles. For the simple case $Q = 4$, we can directly use the thresholds $\{-\mu_Y, 0, \mu_Y\}$. For larger Q, the quantiles will be evenly subdivided. As an example, Figure 5b shows the thresholds for $Q = 8$. Again, since we only need to divide μ_Y by a power of two

(by $\log_2(Q) - 2$ to be precise), the distance of the thresholds can be computed in hardware by right shifting the absolute mean by the required amount. Note that to optimize resource utilization, the individual thresholds are not stored, but will be recalculated on the fly during the quantization, as discussed in the next section.

Response calculation by applying NMQ The NMQ function implements a Non-Monotonic Quantization on raw responses using threshold comparisons. To ease the computation, we can reformulate the definition of NMQ as follows:

$$\mathrm{NMQ}(\delta_c, Q) = \begin{cases} 0, & \text{if}, \delta_c \leq Th_1 \\ 1, & \text{if } Th_i < \delta_c \leq Th_{i+1} \text{ and } i \text{ is even} \\ 0, & \text{if } Th_i < \delta_c \leq Th_{i+1} \text{ and } i \text{ is odd} \\ 1, & \text{if } \delta_c > Th_{Q-1} \end{cases} \tag{6}$$

for some $i \in \{1, \ldots, Q-2\}$, where

- δ_c is the raw response to quantize
- $Q \in \{4, 8, 16, 32\}$ is the quantization level
- Th_i are the $Q-1$ thresholds.

Algorithm 3 Non-Monotonic Quantization (NMQ)

Input:
Raw response δ_c, quantization level $Q \in \{4, 8, 16, 32\}$, absolute mean μ_Y
Output:
NMQ response $resp \in \{0, 1\}$
FUNCTION: $\mathrm{NMQ}(\delta_c, Q, \mu_Y)$
$d \leftarrow \log_2(Q \gg 2)$ {divisor}
step_size $\leftarrow \mu_Y \gg d$
threshold $\leftarrow$ step_size
if $\delta_c < 0$ **then**
  $resp \leftarrow 1$ {inverted polarity for negative δ_c}
else
  $resp \leftarrow 0$
end if
for $i = 1$ **to** $Q - 1$ **do**
  if $|\delta_c| \leq$ threshold **then**
    return $resp$
  end if
  threshold $\leftarrow$ threshold + step_size
  $resp \leftarrow \neg resp$
end for
return $resp$ {highest quantile}

Algorithm 3 shows the pseudo code for our hardware implementation of NMQ, which combines the on-the-fly computation of the thresholds with quantization. It first computes the *step size*, which is the distance between two adjacent thresholds (line 9). It then compares the raw response δ_c to the recomputed thresholds in increasing order. The negative thresholds can be obtained from the positive ones by simply changing the sign. Furthermore, the response for the quantiles in the negative output region is inverted with respect to the corresponding positive quantiles. We can therefore speed up the quantization by only considering the positive thresholds and comparing with the absolute value of δ_c. The opposite polarity of the response for negative δ_c is taken into account in lines 11–15. If δ_c exceeds all thresholds, it returns 1, which is the response corresponding to the highest quantile.

5 Virtual PUF validation

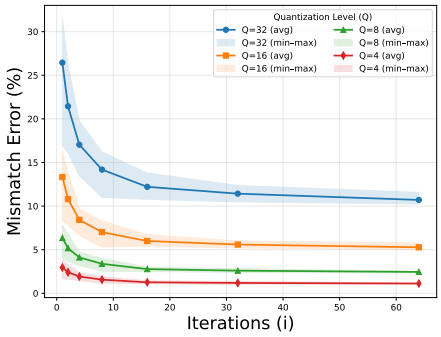
The Virtual PUF concept has been implemented with a Loop PUF composed of $n = 64$ delay elements on an Artix 7 (XC7A35T) FPGA on a Basys 3 development board. In the experiments, we used seven boards of varying age and performance. As a model is used, the reliability is perfect and security is as shown in chapter 3.2. Hence, the validation mainly consists of studying the mismatch error on the different devices and its impact on the authentication protocol for various configurations. The practicality of the Virtual PUF is assessed by verifying its complexity and latency.

5.1 Mismatch Error

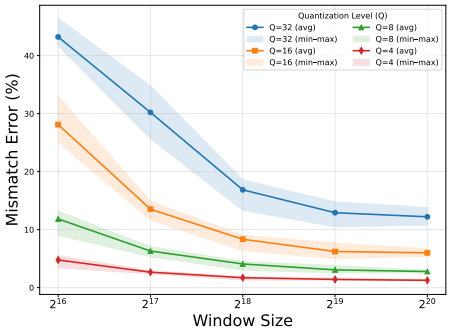
As the PUF uses a digital model during inference, it is perfectly reliable and NMQ therefore greatly enhances security. However, the model built by the Virtual PUF model may differ from those enrolled by the server. This creates a **mismatch error** which corresponds to a false response from the PUF. The main factors that could create mismatch errors are:

- The **quantization level** Q. Higher Q involves more security but also more thresholds and more sensitivity to noise when building the model.
- The **number of iterations** i. Measurement noise depends on the environment, notably the temperature. It can be mitigated by extending the time to build the model by repeating i times the measurements.
- The **window size** w. A larger w reduces measurement noise during model construction and thus reduces mismatch, at the cost of increased start-up latency.
- The **precision of computation**. The delay model may have limited precision to reduce the complexity of the model. Each delay is of fixed-point type $p = x.y$ with x the integer part and y the fractional part.
- The **temperature** T. Temperature variations increase measurement noise, which can lead to a mismatch between the device model and the server model.

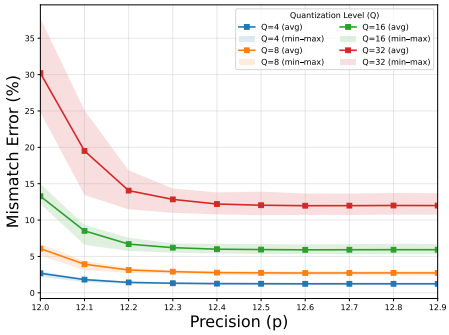
We assessed the mismatch error (e) for different iterations (i), precision (p), temperatures (T), window size (w) and quantization level (Q) by using $n_{ch} = 1000$ random challenges on several PUFs.



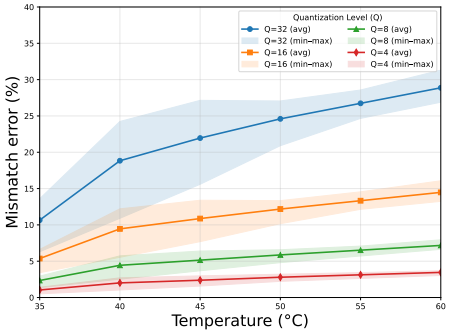
(a) Impact of **iterations** on mismatch error ($p = 12.4$, $T = 32°C$, $w = 2^{20}$)



(b) Impact of **window size** on mismatch error ($p = 12.4$, $T = 29°C$, $i = 16$)



(c) Impact of **precision** on mismatch error ($i = 16$, $T = 32°C$, $w = 2^{20}$)



(d) Impact of **temperature** on mismatch error ($p = 12.4$, $i = 16$, $w = 2^{20}$)

Fig. 6: Average mismatch error rate according to Q for seven devices and other parameters, $n_{ch} = 1000$.

Impact of the number of iterations *i*: The impact of noise can be reduced by using many iterations i when building the model if the noise is independent. We measured the mismatch error rate for $i = 2^k$ with $k \in \{0, ..., 6\}$ and $Q \in \{4, 8, 16, 32\}$. In Figure 6a, we observe that the average mismatch error rate for all Q decreases as the number of iterations increases from $i = 1$ to 16, and stabilizes for larger i. We can see that for this configuration with $w = 2^{20}$ we can use $i = 16$, but a different window size should give different results. Also, the best performing devices can give a mismatch error as low as 10% at $Q = 32$, whereas the worst performing devices can go up to 14%.

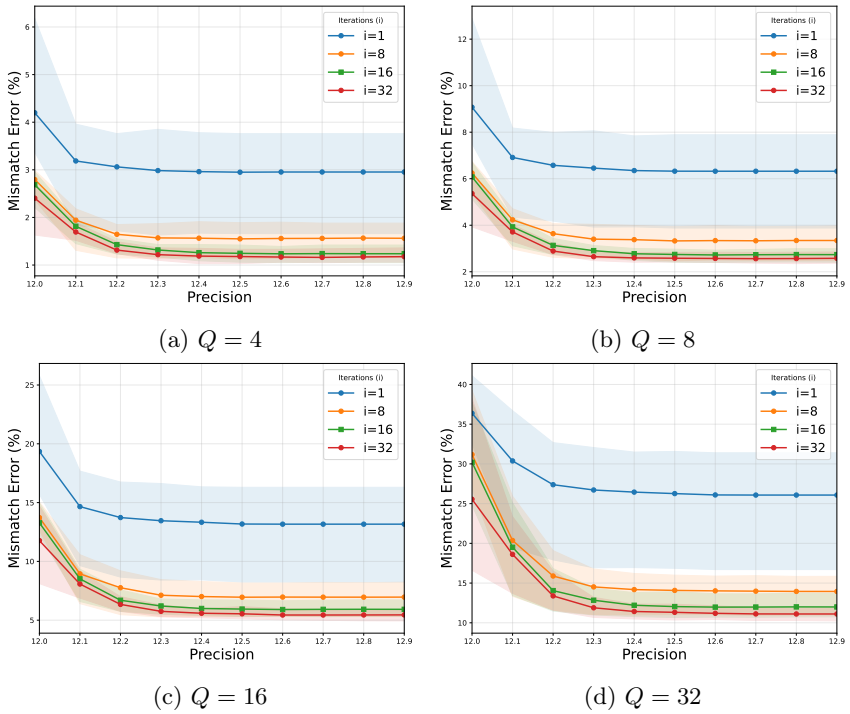
Impact of the window size w: As the number of iterations i increase, a bigger window size w should decrease the noise, but there could be side effects when w is too small or too big. Indeed, when w is small, the PUF output δ_c can be impaired by quantization; and when w is big, the low frequency noise, or flicker noise, can create a bias between the sequential measurements of the challenge c then its complementary $\bar{c}$. Figure 6b shows the results for a constant i=16 and different quantization levels Q. We can see that the average mismatch error and the range of mismatch error over the seven devices decreases within the range $w \in \{2^{16}, 2^{20}\}$.

Impact of the precision p: With $w = 2^{20}$ and $i = 16$, the integer part of δ_c can be represented by 12 bits. We have considered different fixed-point precisions for the models of the n delays, starting from 12 bits by always truncating the fractional part to 12.x bits where we keep x bits of the fractional part. Figure 6c illustrates the effect of model precision on the average mismatch error for $i = 16$, $w = 2^{20}$ and different quantization levels Q using several PUFs. We observe that increasing the precision does not affect all quantization levels in the same way. For $Q = 4$, the curves are almost flat, indicating that the average mismatch error is insensitive to the model precision. In contrast, for higher quantization levels such as $Q = 32$, the average mismatch error drops by more than 50% when a single fractional bit is added, and reaches a reduction of about 75% at $p = 12.4$ bits. Beyond $P = 12.4$ bits, the curves become nearly flat, meaning that further increase in precision only yields negligible improvements.

If we combine the effect of precision p and i for different Q, still for 1000 random challenges, we obtained the results illustrated in Figure 7.

Focusing on Figure 7d for $Q = 32$, the curve for $i = 1$ exhibits the highest average mismatch error and only decreases from approximately 36% to 26% as the precision is increased, indicating that increasing the precision alone is not sufficient: part of the average mismatch error is due to noise and can only be reduced by increasing the number of iterations. In contrast, for $i = 16$, the average mismatch error drops from about 30% to 12% when both the number of iterations and the precision are increased, highlighting the strong combined effect of these two parameters. It is also important to note that there is negligible improvement when the iterations are increased beyond $i = 16$ or the precision beyond 12.4 bits, as the corresponding curves nearly overlap and the residual mismatch error saturates. Across all Q at $i = 1$, there is a large spread between the best and worst performing devices, indicating device-to-device variability. This may be the effect of noise due to device aging. Increasing the number of iterations reduces this variability by averaging out noise, thereby narrowing the gap between the best and worst devices. Therefore, selecting i sufficiently large is important to ensure robust performance across devices with different aging conditions. This confirms that a configuration with $i = 16$ and $p = 12.4$ provides an efficient operating point.

Impact of the temperature T: The server and the PUF may have different temperatures when their respective model is built, but the NMQ thresholds are

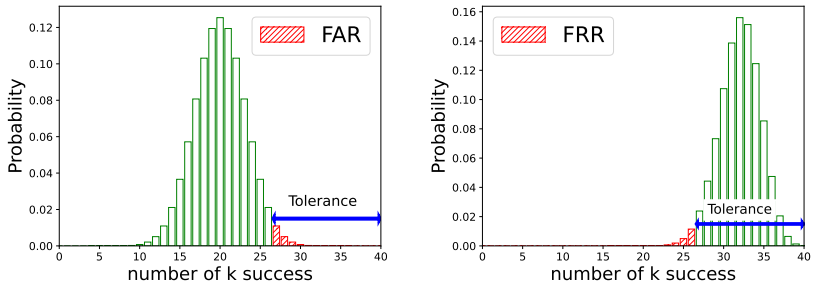


(a) $Q = 4$ (b) $Q = 8$

(c) $Q = 16$ (d) $Q = 32$

Fig. 7: The effect of precision on average mismatch error with different iterations ($T = 32°C$, $w = 2^{20}$).

updated accordingly. From this quantization aspect, we could think that the temperature does not greatly impact the mismatch error. However, as the thermal noise significantly increases with T, it can be the main cause of mismatch error. To evaluate the impact of temperature on the mismatch error, the FPGA board (Basys3) was placed in a Binder KMF series constant climate chamber. The chamber was used to control and stabilize the ambient temperature such that the internal device temperature covered the range from 35°C to 60°C in steps of 5°C for all measurements. Because of self-heating, the device temperature was higher than the ambient temperature inside the Binder KMF chamber; for example, a device temperature of 35°C was obtained with a chamber set-point of approximately 12°C. The actual device temperature was monitored in real time using Xilinx Vivado, which reads the on-chip temperature sensor of the Basys3 FPGA board. Figure 6d shows the influence of the temperature on the average mismatch error for $Q \in \{4, 8, 16, 32\}$ using seven devices. In this experiment, the enrollment temperature is 35°C and the used precision is 12.4, using 1000 challenges for each measurement. Two main trends can be observed. First, for any fixed temperature, the mismatch error increases with the quantization level: at 35°C, it rises from 1.0% for $Q = 4$ to 2.3%, 5.3%, and 10.6% for $Q = \{8, 16, 32\}$, respectively. This reflects the fact that a higher quantiza-



(a) Adversary distribution (b) Legitimate distribution

Fig. 8: Binomial distributions to compute FAR and FRR

tion level makes the threshold boundaries more sensitive to noise and parameter drifts. Second, for any fixed Q, the mismatch error generally increases as the difference between enrollment and device temperature increases. Over the range from 35°C to 60°C, the average mismatch error approximately doubles for all quantization levels: for example, it grows from 1.0% to 3.4% for $Q = 4$, from 5.3% to 14.4% for $Q = 16$, and from 10.6% to 29.0% for $Q = 32$. The increase is roughly linear with temperature, with an average slope between about 0.1 percentage points per °C for $Q = 4$ and 0.6 percentage points per °C for $Q = 32$. Small deviations can be attributed to measurement noise.

5.2 Impact of Mismatch Error on the Authentication

A secure and reliable authentication mainly depends on the **average mismatch error** e and two important parameters that allow the server to adapt the quality of authentication: the **number of challenges** n_{ch} used for authentication and the **tolerance factor** t which tolerates errors on a maximum of t faulty responses among the n_{ch} challenges.

We assessed the False Authentication rate and the False Rejection rates according to these two parameters using seven devices and by considering that all the challenges have the same probability of error.

The False Authentication Rate FAR is computed from the binomial law of the adversary, which has a probability $p = 1/2$. A false authentication occurs when the adversary gets more than $t \cdot n_{ch}$ successes corresponding to the tolerance area, which is in red in Figure 8a. The False Rejection Rate FRR is computed from the binomial law of the legitimate user, which has a probability of the opposite of the mismatch error, i.e., $(1 - e)$. A missed authentication occurs outside the tolerance area when there are fewer than $t \cdot n_{ch}$ success as shown in the red area of Figure 8b.

Figure 9 represents the FAR for different tolerance factors and the number of authentication challenges. We can see that the number of challenges has to be at least 80 to have a FAR of less than 10^{-7}, with a tolerance factor of 20%.

Figure 10 shows the results of FRR for 4 values of average mismatch error rate e corresponding to most cases obtained in subsection 5.1. We can see that the FRR significantly increases with the average mismatch error and the tolerance factor, but this can be compensated for with an increase in the number of challenges.

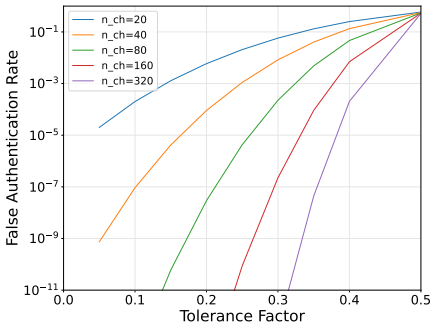


Fig. 9: False authentication rate versus the tolerance parameter t according to different numbers of challenges.

For any configuration, the number of challenges and the tolerance factor can be adjusted to get the required FRR and FAR. For instance, to obtain an FRR and FAR of 10^{-7} with an average mismatch error of 12.2%(refer to Figure 10c), at least 160 challenges are required with a tolerance factor of 25–31% across seven devices.

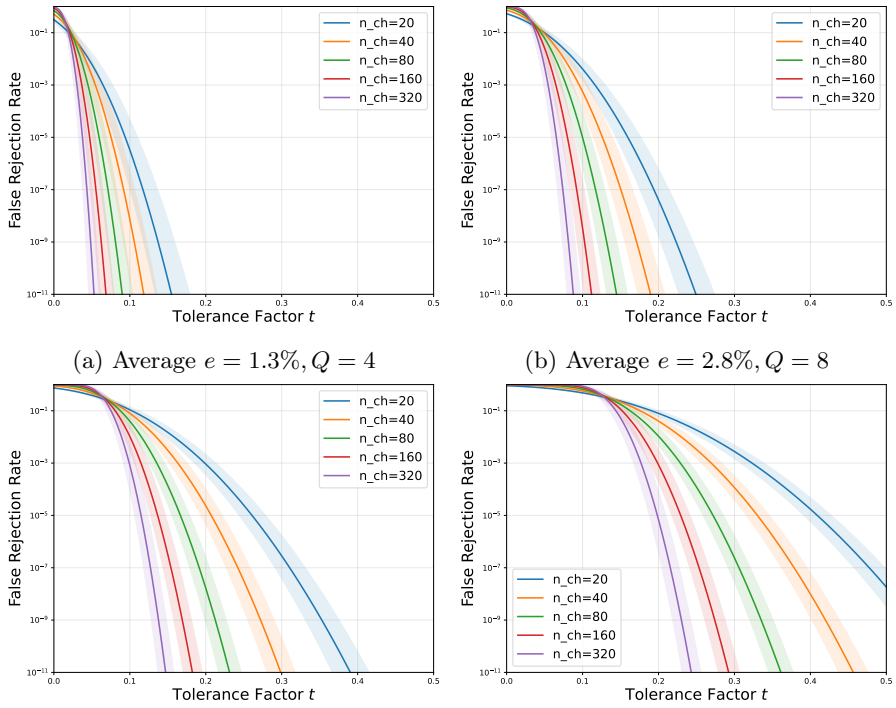
The protocol could be enhanced by taking advantage of statistical properties, such as the Neyman-Pearson test [10] which considers the individual mismatch error rate for each response, or the selection of authentication challenges less prone to mismatch errors.

5.3 Physical Characteristics

Besides its interesting properties of reliability and security, the practicality of the Virtual PUF has to be assessed by verifying its latency and complexity.

Complexity: The complexity of the model depends on parameters such as the length of the PUF n and the precision p. The precision depends on the number of iterations i and the window size w. From Figure 7, we observe that a precision of $p = 12.4$ bits yields the lowest mismatch error for $Q = 32$. This indicates that the first 4 bits of the fractional part are sufficient to build an accurate model.

Table 1 shows the Virtual PUF resource utilization for $n = 64$, $p = 12.4$, $i = 16$, $w = 2^{20}$ and an implemention in a Digilent Basys3 board featuring an Artix-7 XC7A35T FPGA. This FPGA provides 5200 slices (20 800 LUTs and 41 600 flip-flops) and 1.8 Mbit of block RAM [4]. Our implementation requires only



(a) Average $e = 1.3\%, Q = 4$ (b) Average $e = 2.8\%, Q = 8$

(c) Average $e = 6.0\%, Q = 16$ (d) Average $e = 12.2\%, Q = 32$

Fig. 10: False rejection rate versus the tolerance parameter according to different average mismatch error rate e and number of challenges n_{ch} when $i = 16$, $w = 2^{20}$, $p = 12.4$ and $T = 32°\text{C}$ for seven devices.

504 LUTs, 507 registers, and 2304 bits of RAM, corresponding to approximately 2.4% of the available LUTs, 1.2% of the flip-flops, and less than 0.2% of the block RAM. The results show that the implementation of a Virtual PUF keeps a reasonable lightweight property to satisfy the low-complexity requirements.

Latency: During the operational phase, the latency $t_{operation}$ is very low as the response is computed from a simple equation 1 with the fixed-point delays D already modeled beforehand. For instance, if 64 delays are stored in a memory and a clock system frequency is 100MHz, the latency should be 640ns to which the NMQ quantization should be added. Hence, the main contributor to the latency is the communication interface between the PUF and the server. The left column of Table 2 indicates the internal latency $t_{operation}$ without taking account of the communication interface.

The latency t_{model} to build the model is much greater. Indeed, the window size w and the number of iterations i reduce the noise impact but induce in return a greater latency to build the model. Table 2 indicates the latency t_{model}

Table 1: FPGA Resource Utilization by Module, $n = 64$, $p = 12.4$.

Module Name	**LUTs**	**Registers**	**RAM (bits)**
Hadamard challenge Generator	29	63	0
LPUF	137	156	0
Model builder	206	172	1280
Virtual LPUF	58	54	0
Delay models	0	0	1024
NMQ	74	62	0
TOTAL	504	507	2304

Table 2: Measured latency for different window sizes and iterations.

Window Size	t_{model} $(i = 8)$	$t_{\text{model}}(i = 16)$	$t_{\text{operation}}$
16	$0.67s$	$1.34s$	$1.3\mu s$
17	$1.34s$	$2.68s$	$1.3\mu s$
18	$2.68s$	$5.37s$	$1.3\mu s$
19	$5.36s$	$10.73s$	$1.3\mu s$
20	$10.74s$	$21.47s$	$1.3\mu s$

for different configurations of our PUF platform. Some work should be carried out to decrease this latency. It could focus on an optimization of the Loop PUF, or a continuous refinement of the model, even during the operational phase.

6 Conclusions

The proposed Virtual PUF concept shows that it is possible to significantly increase the security of multi-bin PUFs against ML attacks by using NMQ with high quantization levels and a mathematical PUF model that is built at power-up. The resulting complexity remains moderate, notably by taking advantage of the use of a Hadamard matrix construction for the delay models and the folded normal distribution for the quantization levels. This results in a practical and secure authentication system well-suited for resource-constrained IoT devices. The mismatch error between the built-in model and the server model is the main limitation of the approach. It can be dealt with at the protocol level by increasing the number of challenges or the tolerance factor for authentication. Future work covers the refinement of the protocol, for instance, by taking advantage of the Neyman-Pearson Lemma, reducing the mismatch error, and latency to build the model.

Acknowledgments. This work was supported in part by the Agence Nationale de la Recherche (ANR) under Grant ANR-20-CYAL-0007 and in part by the German Federal Ministry of Education and Research (BMBF) within the project APRIORI under Grant 16KIS1389K.

References

1. Becker, G.T.: The gap between promise and reality: On the insecurity of xor arbiter pufs. In: Cryptographic Hardware and Embedded Systems–CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings 17. pp. 535–555. Springer (2015)
2. Cherif, Z., Danger, J.L., Guilley, S., Bossuet, L.: An Easy-to-Design PUF Based on a Single Oscillator: The Loop PUF. In: 2012 15th Euromicro Conference on Digital System Design. pp. 156–162. IEEE, Cesme, Izmir, Turkey (Sep 2012), `http://ieeexplore.ieee.org/document/6386887/`
3. Danger, J.L.: Pufs: Standardization and evaluation. In: 2016 Mobile System Technologies Workshop (MST). pp. 12–18. IEEE (2016)
4. Digilent Inc.: Basys 3™ FPGA Board Reference Manual (2014), `https://digilent.com/reference/programmable-logic/basys-3/reference-manual`, revision 8/22/2014
5. Gassend, B., Lim, D., Clarke, D., Van Dijk, M., Devadas, S.: Identification and authentication of integrated circuits. Concurrency and Computation: Practice and Experience **16**(11), 1077–1098 (2004)
6. Immler, V., Hiller, M., Liu, Q., Lenz, A., Wachter-Zeh, A.: Variable-length bit mapping and error-correcting codes for higher-order alphabet pufs—extended version. Journal of Hardware and Systems Security **3**, 78–93 (2019)
7. Lee, J.W., Lim, D., Gassend, B., Suh, G.E., Van Dijk, M., Devadas, S.: A technique to build a secret key in integrated circuits for identification and authentication applications. In: 2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No. 04CH37525). pp. 176–179. IEEE (2004)
8. Lim, D., Lee, J.W., Gassend, B., Suh, G.E., Van Dijk, M., Devadas, S.: Extracting secret keys from integrated circuits. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **13**(10), 1200–1205 (2005)
9. Maes, R., Verbauwhede, I.: Physically unclonable functions: A study on the state of the art and future research directions. Towards Hardware-Intrinsic Security: Foundations and Practice pp. 3–37 (2010)
10. Nasir, N., Béguinot, J., Cheng, W., Kühne, U., Danger, J.L.: Robust and reliable PUF protocol exploiting non-monotonic quantization and Neyman-Pearson lemma. In: International Conference on Constructive Approaches for Security Analysis and Design of Embedded Systems (CASCADE). pp. 387–409. Springer (2025)
11. Nguyen, P.H., Sahoo, D.P., Jin, C., Mahmood, K., Rührmair, U., Van Dijk, M.: The interpose puf: Secure puf design against state-of-the-art machine learning attacks. Cryptology ePrint Archive (2018)
12. Rioul, O., Solé, P., Guilley, S., Danger, J.L.: On the entropy of physically unclonable functions. In: 2016 IEEE International Symposium on Information Theory (ISIT). pp. 2928–2932. IEEE (2016)
13. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling attacks on physical unclonable functions. In: Proceedings of the 17th ACM conference on Computer and communications security. pp. 237–249. ACM,

Chicago Illinois USA (Oct 2010). https://doi.org/10.1145/1866307.1866335, https://dl.acm.org/doi/10.1145/1866307.1866335
14. Schaub, A., Danger, J.L., Guilley, S., Rioul, O.: An Improved Analysis of Reliability and Entropy for Delay PUFs. In: 2018 21st Euromicro Conference on Digital System Design (DSD). pp. 553–560 (Aug 2018). https://doi.org/10.1109/DSD.2018.00096, https://ieeexplore.ieee.org/document/8491867
15. Solé, P., Cheng, W., Guilley, S., Rioul, O.: Bent sequences over hadamard codes for physically unclonable functions. In: IEEE International Symposium on Information Theory, ISIT 2021, Melbourne, Australia, July 12-20, 2021. pp. 801–806. IEEE (2021). https://doi.org/10.1109/ISIT45174.2021.9517752, https://doi.org/10.1109/ISIT45174.2021.9517752
16. Stangherlin, K., Wu, Z., Patel, H., Sachdev, M.: Enhancing Strong PUF Security With Nonmonotonic Response Quantization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **31**(1), 55–64 (Jan 2023), conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems
17. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th annual Design Automation Conference. pp. 9–14. DAC '07, Association for Computing Machinery, New York, NY, USA (Jun 2007). https://doi.org/10.1145/1278480.1278484, https://dl.acm.org/doi/10.1145/1278480.1278484
18. Tobisch, J., Aghaie, A., Becker, G.T.: Combining optimization objectives: New modeling attacks on strong pufs. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 357–389 (2021)
19. Wisiol, N., Mühl, C., Pirnay, N., Nguyen, P.H., Margraf, M., Seifert, J.P., Van Dijk, M., Rührmair, U.: Splitting the interpose puf: A novel modeling attack strategy. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 97–120 (2020)