Lattice PUF: A Strong Physical Unclonable Function Provably Secure against Machine Learning Attacks

Ye Wang, Xiaodan Xi, and Michael Orshansky
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX, USA
{lhywang, paul.xiaodan, orshansky}@utexas.edu

***Abstract*—We propose a strong physical unclonable function (PUF) provably secure against machine learning (ML) attacks with both classical and quantum computers. Its security is derived from cryptographic hardness of learning decryption functions of public-key cryptosystems. Our design compactly realizes the decryption function of the learning-with-errors (LWE) cryptosystem. Due to the fundamental connection of LWE to lattice problems, we call the construction the lattice PUF.**

Lattice PUF is constructed using a physically obfuscated key (POK), an LWE decryption function block, and a linear-feedback shift register (LFSR) as a pseudo-random number generator. The POK provides the secret key of the LWE decryption function; its stability is ensured by a fuzzy extractor (FE). To reduce the challenge size, we exploit distributional relaxations of space-efficient LWEs. That allows only a small challenge-seed to be transmitted with the full-length challenge generated by the LFSR, resulting in a 100X reduction of communication cost. To prevent an active challenge-manipulation attack, a self-incrementing counter is embedded into the challenge seed.

We prototyped the lattice PUF with 2^{136} challenge-response pairs (CRPs) on a Spartan 6 FPGA, which required 45 slices for the PUF logic proper and 233 slices for the FE. Simulation-based evaluation shows the mean (std) of uniformity to be 49.98% (1.58%), of uniqueness to be 50.00% (1.58%), and of reliability to be 1.26% (2.88%). The LWE concrete hardness estimator guarantees that a successful ML attack of the lattice PUF will require the infeasible 2^{128} CPU operations. Several classes of empirical ML attacks, including support vector machine, logistic regression, and deep neural networks, are used: in all attacks, the prediction error remains above 49.76% after 1 million training CRPs.

***Index Terms*—Strong PUF, PAC Learning, Lattice Cryptography, ML Resistance.**

I. Introduction

Silicon physical unclonable functions (PUFs) are security primitives widely used in device identification, authentication, and cryptographic key generation [45]. Given an input challenge, a PUF exploits the randomness inherent in CMOS technology to generate an output response. In contrast to weak PUFs, also called physically obfuscated keys (POKs) using the taxonomy of [19], which supply limited amount of challenge-response pairs (CRPs), strong PUFs have an exponentially large CRP space.

In this paper, we propose a strong PUF that is secure against machine learning (ML) attacks with both classical and quantum computers. As a formal framework to define ML resistance, we adopt the probably approximately correct (PAC) theory of learning [38]. Specifically, the PAC non-learnability of a decryption function implies that with a polynomial number of samples, with high probability, *it is not possible to learn a function accurately by any means.* The main insight, which allows us to build such a novel strong PUF, is our reliance on the earlier proof that PAC-learning a decryption function of a semantically secure public-key cryptosystem entails breaking that cryptosystem [25], [26], [28]. We develop a PUF for which the task of modeling is equivalent to PAC-learning the decryption function of a learning-with-errors (LWE) public-key cryptosystem. The security of LWE cryptosystems is based on the hardness of LWE problem that ultimately is reduced to the hardness of several problems on lattices [41]. The input-output mapping between the PUF and the underlying LWE cryptosystem can be briefly summarized as follows: challenge $\Longleftrightarrow$ ciphertext and response $\Longleftrightarrow$ decrypted plaintext. Notably, LWE is believed to be secure against both classical and quantum computers. Because of the intrinsic relation between the proposed PUF and the security of lattice cryptography we call our construction the **lattice PUF**.

The lattice PUF is constructed using a POK, an LWE decryption function block, a linear-feedback shift register (LFSR), a self-incrementing counter, and a control block. The entire implementation is lightweight and fully digital.

The LWE decryption function block is the core module of the lattice PUF, generating response (plaintext) to each submitted challenge (ciphertext). Design parameters of the LWE decryption function in the lattice PUF are chosen by balancing the implementation costs, statistical performance, and the concrete hardness of ML resistance. We develop

a measure of ML security in terms of the total number of operations needed to learn a model of the PUF. Such concrete hardness is established by the analysis of state-of-the-art attacks on the LWE cryptosystem [31], [37] and evaluated by the estimator developed by Albrecht et al. [3]. Using this estimator, we say that a PUF has k-bit ML resistance if a successful ML attack requires 2^k operations. We implement the LWE decryption function with guaranteeing 128-bit ML resistance. However, directly using a LWE decryption function as a strong PUF is not efficient since 1-bit of response requires 1288-bit input challenges.

We further develop an improved design for resource-constrained environments that dramatically (by about 100X) reduces the communication cost associated with PUF response generation. This is achieved by *by exploiting distributional relaxations allowed by recent work in space-efficient LWEs* [15]. This allows introducing a low-cost pseudo-random number generator (PRNG) based on an LFSR and transmitting only a small seed. Finally, while the focus of the paper is a PUF that is secure against passive attacks, we address the risk of an active attack by adopting the technique in [49]: we introduce a self-incrementing counter and embed the counter value into a challenge seed. This makes the attack impossible as the counter restricts the attacker's ability to completely control input challenges to the LWE decryption function.

We construct the lattice PUF to achieve a CRP space of size 2^{136}. Statistical simulation shows excellent uniformity, uniqueness, and reliability of the proposed lattice PUF. The mean (standard deviation) of uniformity is 49.98% (1.58%), and of inter-class HD is 50.00% (1.58%). The mean BER (intra-class Hamming distance (HD)) is 1.26%. We also validate the empirical ML resistance of the constructed lattice PUF via support vector machines (SVM), logistic regression (LR), and neural networks (NN). Even with a deep neural network (DNN), which is considered to be one of the most powerful and successful ML attacks today, the prediction error stays above 48.81% with 1 million training samples. The proposed lattice PUF requires a 1280-bit secret key. A concatenated-code-based fuzzy extractor (FE) is utilized to reconstruct stable POK bits. Assuming an average bit error rate (BER) of 5% for raw SRAM cells, the total number of raw SRAM bits needed is 6.5K, in order to achieve a key reconstruction failure rate of 10^{-6}. We implement the entire PUF system (except for raw SRAM cells) on a Spartan 6 FPGA. The PUF logic, including an LWE decryption function, a 256-tap LFSR, a 128-bit self-incrementing counter, requires only 45 slices. The concatenation-code-based FE takes 233 slices. Compared to several known strong PUFs, the proposed PUF is significantly more resource-efficient.

II. Background Work

In order for a strong PUF to be an effective security primitive, the associated CRPs need to be unpredictable. In other words, strong PUFs are required to be resilient to modeling attacks via ML. The question of whether it is possible to engineer a ML secure and lightweight strong PUF has been a long-lasting challenge [46].

SVM is utilized to successfully attack a 64-bit arbiter PUF (APUF) in [30]. Subsequent modification of the original APUF aimed to strengthen ML resistance, including bistable ring PUF [10], feed-forward APUF [30], and lightweight secure PUF (LSPUF) [36], have also been broken via improved ML attacks [5], [42], [44], [16]. Recent proposed interpose PUF (IPUF) [39] claims provable ML resistance by assuming XOR APUFs are hard to learn and rigorously reducing IPUF modeling to XOR APUFs. Unfortunately, both their assumption and claims are proved wrong: [43] demonstrates that XOR APUFs and IPUFs are actually vulnerable to deep-learning-based modeling attacks. There are often complex reasons why claims and rigorous proofs of security fail in practice. The most fundamental one is that their claims rely on recent conjectures made from empirical findings. In contrast, the security proof of lattice PUF is based on the hardness of several basic lattice problems, which are seen as foundational results in math and computer science, and are widely believed true.

By exploiting higher intrinsic nonlinearity, some strong PUFs [29], [47] exhibit empirically-demonstrated resistance to a list of ML algorithms. Empirical demonstrations of ML resistance are not fully satisfactory since they can never rule out the possibility of other more effective ML algorithms.

The so-called controlled PUF setting [17] attempts to ensure the ML resistance via cryptographic primitives such as hash functions. However, the use of hash functions inside a PUF endangers the promise of a strong PUF as a lightweight structure. Strong PUF constructions using established cryptographic ciphers, such as AES [7], have similar challenges.

Recent work [14], [19], [21] have also utilized lattice-based problems, including learning-parity-with-noise (LPN) and LWE, to realize computationally-secure FEs and, as a byproduct, to construct strong PUFs. [1] The fundamental security property that [14], [19], [21] rely upon is the *computational hardness of recovering a private key from a public key in a public-key cryptosystem*. Their CRP generation is based on generating multiple private keys (playing the role of PUF responses) and multiple public keys (playing the role of PUF challenges). This is only possible because multiple public keys are derived using a fixed (same) source of secret POK bits, embedded in the error term of LPN or LWE. As was shown in [4], the fact that multiple CRPs have shared error terms can be easily exploited allows a computationally-inexpensive algorithm for solving an LPN or LWE instance, thus compromising the hardness of LPN or LWE problems. Thus, by itself [14], [19], [21], *the resulting PUF does not have resistance against ML modeling attacks*. This vulnerability is fixed in [19], [21] by introducing a cryptographic hash function to hide the original CRPs, which violate the principle of lightweightness. *In stark contrast, the*

[1] A computational FE guarantees absence of information leakage from publicly shared helper data via computational hardness in contrast to conventional FEs that need to limit their information-theoretic entropy leakage.

proposed lattice PUF derives its security by directly exploiting a distinctly different property of public-key cryptosystems: the theoretically-proven guarantee that their decryption functions are not PAC-learnable. In the lattice PUF, the above-discussed vulnerability is absent since the publicly known challenges are ciphertexts and the security of the cryptosystem ensures that a fixed private key (the POK, in our case) cannot be recovered from ciphertexts.

III. LWE Decryption Functions Are Hard to Learn

This section formally defines ML resistance of strong PUFs via the notion of PAC learning and shows why LWE decryption functions are attractive for constructing post-quantum ML-resistant PUFs. In this section, we focus on passive attacks in which the attacker can observe the challenges sent to the verifier but is unable to generate challenges of his or her choice.

A. ML Resistance as Hardness of PAC Learning

A strong PUF can be modeled as a function $f : \mathcal{C} \to \mathcal{R}$ mapping from the challenge space $\mathcal{C}$ (usually $\{0,1\}^n$) to the response space $\mathcal{R}$ (usually $\{0,1\}$). We call f the true model of a strong PUF since it captures the exact challenge-response behavior.

ML attacks are usually performed by relying on a functional class of candidate models, collecting CRPs as the training data, and running a learning algorithm to obtain a model from the candidate class which best approximates the true model. In addition to the approximation quality, the criteria of evaluating the effectiveness and efficiency of the learning algorithm also include the sample and time complexity. To claim that a strong PUF is easy to learn, one can propose a learning algorithm which finds a CRP model with good approximation quality using a small number of sample CRPs and terminates in a short time. The converse is difficult: to claim that a PUF is hard to learn, one must show that all possible learning algorithms fail to provide models with good approximation quality, or they require a large number of CRPs or a long running time.

We argue that the only known framework for seeking a provable notion of ML resistance with a formal analysis of approximation quality, sample size, and time complexity is the PAC learning model [38]. We now formalize the passive modeling attack scenario in the context of PAC learning. A PAC-term for a true model f of a strong PUF is a concept. Denote as $\mathcal{F}$ the set of all possible PUF-realized functions (every instance of a PUF creates its unique functional mapping f). The set of candidate models used in the learning algorithm is the hypothesis set $\mathcal{H}$. The goal of a learning algorithm is to select a candidate model that matches the true model well. Importantly, as shown later, the proof of PAC-hardness guarantees that $\mathcal{H}$ does not have to be restricted to be the same as $\mathcal{F}$ of true models. This generalization permits a stronger *representation-independent* PAC-hardness proof. While not always possible, representation-independent hardness can be proven for PAC-learning of decryption functions ensuring that no matter how powerful and expressive the chosen $\mathcal{H}$ is, PAC learning decryption function requires exponential time.

Within the PAC model, CRPs in a training set are assumed to be independent and identically distributed (i.i.d.) under a certain distribution $\mathcal{D}$.

We say a set $\mathcal{F}$ of strong PUFs is PAC-learnable using $\mathcal{H}$, if there exists a polynomial-time algorithm $\mathcal{A}$ such that $\forall \epsilon > 0$, $\forall \delta > 0$, for any fixed CRP distribution $\mathcal{D}$, and $\forall f \in \mathcal{F}$, given a training set of size m, $\mathcal{A}$ produces a candidate model $h \in \mathcal{H}$ with probability of, at least, $1 - \delta$ such that

$$\Pr_{(\mathbf{c},r)\sim\mathcal{D}}[f(\mathbf{c}) \neq h(\mathbf{c})] < \epsilon.$$

In conclusion, our strategy is to say that a strong PUF is ML-resistant if it is not PAC-learnable (i.e., that it is PAC-hard). PAC-hardness implies that any successful ML attack requires at least an exponential running time.

B. Decryption Functions Are not PAC Learnable

What is critically important is that there exist functions that are known to be not PAC-learnable. Specifically, a class of decryption functions of secure public-key cryptosystems is not PAC-learnable, as established by [25], [28]. We outline their proof below.

A public-key cryptosystem is a triple of probabilistic polynomial-time algorithms $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ such that: (1) Gen takes n as a security parameter and outputs a pair of keys (pk, sk), the public and private keys respectively; (2) Enc takes as input the public key pk and encrypts a message (plaintext) r to return a ciphertext $\mathbf{c} = \mathrm{Enc}(pk, r)$; (3) Dec takes as input the private key sk and a ciphertext $\mathbf{c}$ to decrypt a message $r = \mathrm{Dec}(sk, \mathbf{c})$. We only need to discuss public-key cryptosystems encrypting 1-bit messages (0 and 1).

One of the security requirements of a public-key cryptosystem is that it is computationally infeasible for an adversary, knowing the public key pk and a ciphertext $\mathbf{c}$, to recover the original message, r. This requirement can also be interpreted as the need for indistinguishability under the chosen plaintext attack (also often referred to as semantic security requirement) [24]. Given the encryption function Enc and the public key pk, the goal of an attacker is to devise a *distinguisher* $\mathcal{A}$ to distinguish between encryption $\mathrm{Enc}(pk, r)$ of $r = 0$ and $r = 1$ with non-negligible probability:

$$|\Pr[\mathcal{A}(pk, \mathrm{Enc}(pk, 0)) = 1] - \Pr[\mathcal{A}(pk, \mathrm{Enc}(pk, 1)) = 1]| \geq \epsilon.$$

A cryptosystem is semantically secure if no polynomial-time attacker can correctly predict the message bit with non-negligible probability.

The connection between the above-stated security of a public-key cryptosystem and the hardness of learning a concept class associated with its decryption function was established in [25], [28]. The insight of [25], [28] is that PAC-learning is a natural result of the ease of encrypting messages with a public key. Since the encryption function Enc and the public-key pk is known, the distinguishing algorithm can sample independent training examples in the following way:

(1) picking a plaintext bit r uniformly randomly from $\{0,1\}$, (2) encrypting r to get the ciphertext $\mathbf{c} = \mathrm{Enc}(pk, r)$. (We later refer to the resulting distribution of ciphertext as the "ciphertext distribution".) Next, the distinguishing algorithm passes the set of training examples $((\mathbf{c}, r)$'s) into an algorithm for learning the decryption function $\mathrm{Dec}(sk, \cdot)$. The PAC learning algorithm returns a model $h(\cdot)$ that aims to approximate $\mathrm{Dec}(sk, \cdot)$. Using $h(\cdot)$, one could distinguish between ciphertexts stemming from $r = 0$ and $r = 1$ with non-negligible probability. This would entail violating the semantic security of the cryptosystem. Technically, this can be summarized as follows [25], [28].

Theorem 1: If a public-key cryptosystem is secure against chosen plaintext attacks, then its decryption functions are not PAC-learnable (under the ciphertext input distribution).

C. LWE Is Post-Quantum Secure

According to the cryptographic hardness above, decryption functions of any secure public-key cryptosystem, such as Rivest–Shamir–Adleman (RSA) and elliptic-curve cryptography (ECC), can be used to construct ML-resistant PUFs. However, integer-factoring-based cryptosystems, including RSA and ECC above, become insecure with the development of quantum computers. Among all post-quantum schemes [6], the LWE cryptosystem based on hard lattice problems appears to be most promising due to its implementation efficiency and stubborn intractability since 1980s.

A lattice $\mathcal{L}(\mathbf{V})$ in n dimensions is the set of all integral linear combinations of a given basis $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ with $\mathbf{v}_i \in \mathbb{R}^n$:

$$\mathcal{L}(\mathbf{V}) = \{a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots a_n\mathbf{v}_n : \forall a_i \in \mathbb{Z}\}.$$

The LWE problem is defined on the integer lattice $\mathcal{L}(\mathbf{V}) = \{(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle)\}$ with a basis $\mathbf{V} = (\mathbf{I}; \mathbf{s})$, in which $\mathbf{I}$ is an n-dimensional identity matrix and $\mathbf{s}$ is a fixed row vector (also called the secret) in $\mathbb{Z}_q^n$. Throughout this paper, vectors and matrices are denoted with bold symbols with dimension on superscript, which can be dropped for convenience in case of no confusion. Unless otherwise specified, all arithmetic operations in the following discussion including additions and multiplications are performed in $\mathbb{Z}_q$, i.e. by modulo q.

For the lattice $\mathcal{L}(\mathbf{V}) = \{(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle)\}$ with dimension n, integer modulus q and a discrete Gaussian distribution $\bar{\Psi}_\alpha$ for noise, the LWE problem is defined as follows. The secret vector $\mathbf{s}$ is fixed by choosing its coordinates uniformly randomly from $\mathbb{Z}_q$. Next $\mathbf{a}_i$'s are generated uniformly from $\mathbb{Z}_q^n$. Together with the error terms e_i, we can compute $b_i = \langle \mathbf{a}, \mathbf{s}\rangle + e_i$. Distribution of $(\mathbf{a}_i, b_i)$'s over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is called the LWE distribution $A_{\mathbf{s},\bar{\Psi}_\alpha}$. The most important property of $A_{\mathbf{s},\bar{\Psi}_\alpha}$ is captured in the following lemma:

Lemma 1: Based on hardness assumptions of several lattice problems, the LWE distribution $A_{\mathbf{s},\bar{\Psi}_\alpha}$ of $(\mathbf{a}, b)$'s is indistinguishable from a uniform distribution in $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Solving the decision version of LWE problem is to distinguish with a non-negligible advantage between samples from $A_{\mathbf{s},\bar{\Psi}_\alpha}$ and those generated uniformly from $\mathbb{Z}_q^n \times \mathbb{Z}_q$. This LWE problem is shown to be intractable to solve, without knowing the secret $\mathbf{s}$, based on the worst-case hardness of several lattice problems [41]. Errors e are generated from a discrete Gaussian distribution $\bar{\Psi}_\alpha$ on $\mathbb{Z}_q$ parameterized by $\alpha > 0$: sampling a continuous Gaussian random variable with mean 0 and standard deviation $\alpha q/\sqrt{2\pi}$ and rounding it to the nearest integer in modulo q. Notice that error terms are also essential for guaranteeing the indistinguishability: without noise $(\mathbf{a}, b)$ becomes deterministic and the secret $\mathbf{s}$ can be solved efficiently via Gaussian elimination methods.

We now describe a public-key cryptosystem based on the LWE problem above in [9]:

Definition 1: (LWE cryptosystem)

- **Private key:** $\mathbf{s}$ is uniformly random in $\mathbb{Z}_q^n$.
- **Public key:** $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ is uniformly random, and $\mathbf{e} \in \mathbb{Z}_q^m$ with each entry from $\bar{\Psi}_\alpha$. Public key is $(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e})$.
- **Encryption:** $\mathbf{x} \in \{0,1\}^m$ is uniformly random. To encrypt a one-bit plaintext r, output ciphertext $\mathbf{c} = (\mathbf{a}, b) = (\mathbf{A}^T\mathbf{x}, \mathbf{b}^T\mathbf{x} + r\lfloor q/2\rfloor)$.
- **Decryption:** Decrypt the ciphertext $(\mathbf{a}, b)$ to 0 if $b - \langle \mathbf{a}, \mathbf{s}\rangle$ is closer to 0 than to $\lfloor q/2\rfloor$ modulo q, and to 1 otherwise.

Notice that each row in the public-key $(\mathbf{A}, \mathbf{b})$ is an instance from the LWE distribution $A_{\mathbf{s},\bar{\Psi}_\alpha}$.

Correctness of the LWE cryptosystem can be easily verified: without the error terms, $b - \langle \mathbf{a}, \mathbf{s}\rangle$ is either 0 or $\lfloor q/2\rfloor$, depending on the encrypted bit. Semantic security of the LWE cryptosystem follows directly from the indistinguishability of the LWE distribution from the uniform distribution in $\mathbb{Z}_q^n \times \mathbb{Z}_q$. Ciphertexts $(\mathbf{a}, b)$ are either linear combinations or shifted linear combination of LWE samples, both of which are indistinguishable from the uniform distribution. This is true because shifting by any fixed length preserves the shape of a distribution. Therefore, an efficient algorithm that can correctly guess the encrypted bit would be able to distinguish LWE samples from uniformly distributed samples. This allows [41] to prove that:

Theorem 2: Based on the hardness assumptions of several lattice problems, the LWE cryptosystem is secure against the chosen-plaintext attacks using both classical and quantum computers.

When the error terms e_i's are introduced:

$$\begin{aligned} b - \langle \mathbf{a}, \mathbf{s}\rangle &= \sum_{i\in S} b_i + \lfloor \frac{q}{2}\rfloor r - \langle \sum_{i\in S} \mathbf{a}_i, \mathbf{s}\rangle \\ &= \sum_{i\in S} (\langle \mathbf{a}_i, \mathbf{s}\rangle + e_i) - \lfloor \frac{q}{2}\rfloor r - \langle \sum_{i\in S} \mathbf{a}_i, \mathbf{s}\rangle \\ &= \lfloor \frac{q}{2}\rfloor r - \sum_{i\in S} e_i, \end{aligned}$$

in which S is the set of non-zero coordinates in $\mathbf{x}$. For a decryption error to occur, the accumulated error $\sum_{i\in S} e_i$ must be greater than the decision threshold $\lfloor q/4\rfloor$. The probability

of the error is given by [37]:

$$\begin{aligned}\text{Err}_{\text{LWE}} &\approx 2(1-\Phi(\frac{q/4}{\alpha q\sqrt{m/2}/\sqrt{2\pi}})) \\ &= 2(1-\Phi(\frac{\sqrt{\pi}}{2\alpha\sqrt{m}})),\end{aligned}$$

in which $\Phi(\cdot)$ is the cumulative distribution function of the standard Gaussian variable. We later use this expression to find the practical parameters for the lattice PUF.

IV. Design of Lattice PUF

The theoretical security guarantees in Section III shows that an LWE decryption function can be used as a strong PUF with challenges generated from a ciphertext distribution. In this section, we first derive design parameters for the LWE decryption function and show that such a direct implementation of lattice PUF is inefficient in resource constrained environments due to high-ratio of ciphertext to plaintext. As we will illustrate in the following, an LWE decryption function with a 128-bit concrete ML hardness requires transmitting $128.8K$ challenge bits in order to produce a 100-bit response string. We then solve this problem *by exploiting distributional relaxations allowed by recent work in space-efficient LWEs*. The proposed strategy allows introducing a low-cost PRNG based on an LFSR and transmitting only a small seed, which results in a dramatic reduction of effective challenge size. Last, we introduce a simple defense to protect our PUF against a standard active attack on the LWE decryption function.

The top-level architecture of the proposed lattice PUF is shown in Figure 1.

A. LWE Decryption Function

Figure 2 shows the core component of the proposed lattice PUF: the LWE decryption function. It takes a binary challenge vector $\mathbf{c} = \{c_0, c_1, \ldots, c_{N-1}\}$ of size $N = (n+1)\log q$ which maps to a ciphertext $(\mathbf{a}, b)$ in the following way:

$$\begin{aligned}a_i &= \sum_{j=0}^{\log q-1} c_{(i-1)\log q+j} 2^j,\ \forall i \in \{1,2,\ldots,n\}, \\ b &= \sum_{j=0}^{\log q-1} c_{n\log q+j} 2^j.\end{aligned}$$

Here a_i denotes the i-th element of the integer vector $\mathbf{a} \in \mathbb{Z}_q^n$. In this paper, without specification, $\log(x)$ refers to $\log_2(x)$. Similarly, the private key $\mathbf{s}$ for the corresponding LWE decryption function is realized by a binary secret key $\mathbf{W} = \{W_0, W_1, \ldots, W_{n\log q-1}\}$ of size $n\log q$:

$$s_i = \sum_{j=0}^{\log q-1} W_{(i-1)\log q+j} 2^j,\ \forall i \in \{1,2,\ldots,n\}.$$

A modulo-dot-product $b - \langle \mathbf{a}, \mathbf{s}\rangle$ is computed using the modulo-multiply-accumulate unit. It can be implemented in a serial way using n stages. Recall that all additions and multiplications are performed in modulo q. Since q is a power of 2 in our construction, modulo addition and multiplication can be naturally implemented by integer addition and multiplication that keep only the last $\log q$-bit result. Finally the response r is produced by a quantization operation $r = Q(b - \langle \mathbf{a}, \mathbf{s}\rangle)$:

$$Q(x) = \begin{cases} 0 & x \in [0, \frac{q}{4}] \cup (\frac{3q}{4}, q-1], \\ 1 & x \in (\frac{q}{4}, \frac{3q}{4}]. \end{cases}$$

The computation above can be directly implemented as a strong PUF with 2^N CRPs since it maps a challenge vector $\mathbf{c} \in \{0,1\}^N$ into a binary response $r \in \{0,1\}$. We now discuss parameter selection for the LWE decryption function. In general, we seek to find design parameters such that (1) the resulting PUF has excellent statistical properties, such as uniformity, uniqueness, and reliability, (2) successful ML attacks against it require an un-affordably high time complexity in practice, and (3) its hardware implementation costs are minimized.

Prior theoretical arguments establish the impossibility of a polynomial-time attacker. To guarantee practical security, we need to estimate the number of samples and the actual running time (or a number of CPU operations) required for a successful ML attack. [41] shows that a small number of samples are enough to solve an LWE problem, but in an exponential time. Thus, we refer to runtime as concrete ML resistance (or ML hardness) and say that a PUF has k-bit ML resistance if any successful ML attack requires at least 2^k operations. We adopt the estimator developed by Albrecht *et al.* [3] to estimate concrete ML hardness. The concrete hardness of an LWE problem increases with the increase of LWE parameters n, q, and α for all types of attacks. Recall that n represents the lattice dimension, q represents the range of integer for each dimension, and α reflects the noise level in CRP (ciphertext) generation. For a given set of parameters, the estimator compares the complexity of several most effective attacks, including decoding, basis reduction, and meet-in-the-middle attacks [11], [20], [31]. We utilize the estimator in a black-box fashion to find the set of parameters with the target of 128-bit concrete ML resistance.

We consider two metrics of implementation cost, both of which scale with n: the number of challenge and secret bits needed ($n\log q$), and the number of multiply-accumulate (MAC) operations (n). This motivates the need to decrease n.

For conventional PUFs, such as APUF and SRAM PUF, an output error is due to environmental noise, e.g. delay changes in APUF and FET strength changes in SRAM PUF with both voltage and temperature. In contrast, output errors of the lattice PUF come from two sources: (1) environmental errors of secret bits, and (2) errors of decryption during response generation. The former can be thought as the failure of key reconstruction in POKs. Since a single bit-flip completely changes the challenge-response behavior of LWE decryption function, the failure rate of key reconstruction needs to be low, e.g. 10^{-6} (as widely adopted in other PUF applications [34]). Section V describes how the target failure rate can be achieved via a conventional FE based on the error-correcting codes. The

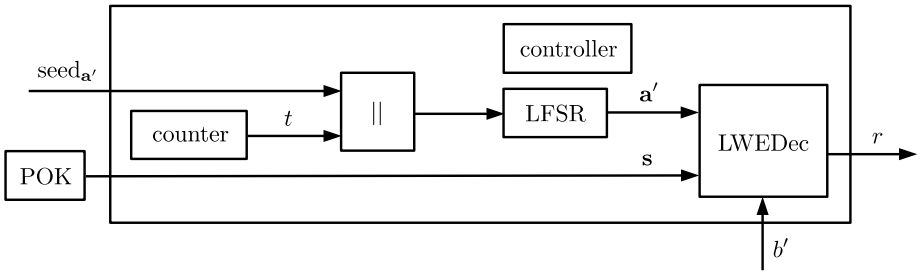

Fig. 1: Top-level architecture and data flow of the lattice PUF.

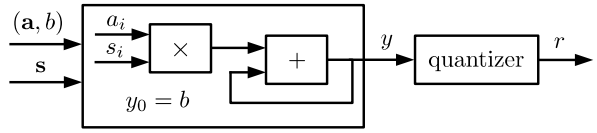

Fig. 2: Architecture of LWE decryption function.

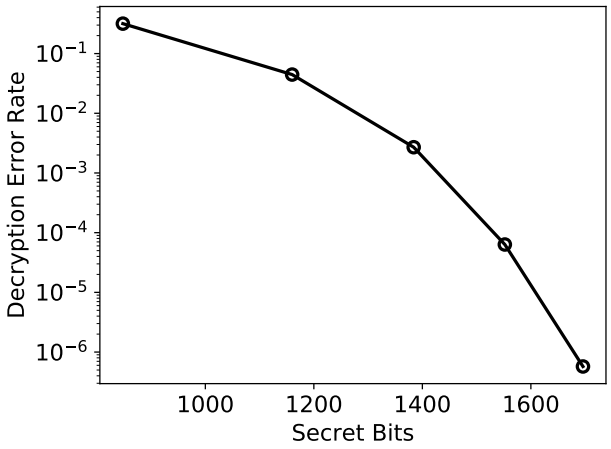

Fig. 3: Super-exponential decrease of decryption error rate with the increase of secret bits. The analysis is done for 128-bit concrete hardness.

latter corresponds to the decryption error and is orthogonal to errors in the secret key $\mathbf{s}$. Recall that in CRP generation of the lattice PUF, a bit of plaintext r is sampled and the ciphertext $\mathbf{c}$ is produced by a noisy encryption function $\mathbf{c} = \mathrm{Enc}(r)$. Given ciphertext $\mathbf{c}$ as input challenge, the decryption function can output a wrong response $r' \neq r$ when the accumulated error $\sum_{i \in S} e_i$ in the encryption function exceeds the decision boundary.

The model for evaluating the decryption error rate is shown in Section II. In order for a strong PUF to be used in direct authentication, its decryption error rate should be small enough for reliable distinguishability of long strings. We set the target around 2%. Figure 3 explores the trade-off between the number of secret bits and the decryption error rate needed for 128-bit concrete ML hardness. It shows that, at fixed concrete ML hardness, the decryption error rate decreases super exponentially with the number of secret bits.

Considering the design metrics above, a feasible set of parameters is found using the estimator in [3]. By setting $n = 160$, $q = 256$, $m = 256$ and $\alpha = 2.20\%$, we achieve a lattice PUF with 128-bit concrete hardness and a decryption error rate of 1.26%.

In order to get a 1-bit response, $(n+1)\log q = 1288$ bits need to be sent to the lattice PUF as a challenge. For direct authentication applications, usually around 100 bits of responses are required. Therefore, the direct implementation described so far would require $C = 128.8K$ challenge bits. This high ratio of challenge length to response length limits its practical use in many scenarios when communication is expensive.

B. Challenge Compression through Distributional Relaxation

We now describe the proposed strategy based on space-efficient LWE that overcomes the limitation on communication inefficiency. The LWE decryption function described in Section IV-A requires a challenge $\mathbf{c}$ in the form $\mathbf{c} = (\mathbf{a}, b)$ to be sent from the server to the PUF. To represent vector $\mathbf{a} \in \mathbb{Z}_q^n$ requires $n \log q$ bits while to represent scalar $b \in \mathbb{Z}_q$ requires only $\log q$ bits. Thus, the major cost of transmission lies in sending $\mathbf{a}$. We wish to avoid sending $\mathbf{a}$ directly and, instead, to send a compressed (shorter) version of $\mathbf{a}$ and re-generate its full-size version on the PUF. Our approach is enabled by the recent results on the distributional behavior of $\mathbf{a} = \mathbf{A}^T\mathbf{x}$ [2] and the concept of space-efficient LWE [15].

Recall that b is given by:

$$\begin{aligned} b &= \mathbf{b}^T\mathbf{x} + r\lfloor q/2 \rfloor \\ &= (\mathbf{A}\mathbf{s} + \mathbf{e})^T\mathbf{x} + r\lfloor q/2 \rfloor \\ &= (\mathbf{A}^T\mathbf{x})^T\mathbf{s} + \mathbf{e}^T\mathbf{x} + r\lfloor q/2 \rfloor. \end{aligned}$$

First, we replace the component $\mathbf{a} = \mathbf{A}^T\mathbf{x}$ by $\mathbf{a}^*$ uniformly randomly sampled from $\mathbf{Z}_q^n$. That allows us to represent challenge $\mathbf{c} = (\mathbf{a}, b)$:

$$\begin{cases} \mathbf{a} = \mathbf{A}^T\mathbf{x} \\ b = (\mathbf{A}^T\mathbf{x})^T\mathbf{s} + \mathbf{e}^T\mathbf{x} + r\lfloor q/2 \rfloor \end{cases}$$

as $\mathbf{c}^* = (\mathbf{a}^*, b^*)$:

$$\begin{cases} \mathbf{a}^* \\ b^* = \mathbf{a}^{*T}\mathbf{s} + \mathbf{e}^T\mathbf{x} + r\lfloor q/2 \rfloor \end{cases}.$$

In [2], it is proven that distribution of $\mathbf{c}^* = (\mathbf{a}^*, b^*)$ is statistically close to the original ciphertext distribution, therefore the required security properties are preserved.

The advantage of the above approximation is that, as shown by [15], several low-complexity PRNGs are capable of producing an output string $\mathbf{a}'$ suitably close to $\mathbf{a}^* \in \mathbb{Z}_q^n$ within the context of LWE cryptosystem. In particular, an LFSR is an especially simple PRNG having the right properties. Specifically, a vector $\mathbf{a}'$ generated by an LFSR provides similar concrete security guarantees against standard attacks on LWE, such as CVP reduction, decoding, and basis reduction [15]. This is because LFSR-generated $\mathbf{a}'$ maintains good properties including:

- it is hard to find "nice" bases for a lattice with basis from LFSR-generated $\mathbf{a}'$;
- given an arbitrary vector in $\mathbb{Z}_q^n$, it is hard to represent it as a binary linear combination of LFSR-generated $\mathbf{a}'$'s;
- it is hard to find a short vector $\mathbf{w}$ that is orthogonal to LFSR-generated $\mathbf{a}'$'s.

The ability to rely on a simple PRNG to produce $\mathbf{a}'$ allows a dramatic reduction in challenge transfer cost. Now, the challenge $\mathbf{c}'$ contains only a small $\text{seed}_{\mathbf{a}'}$ into the PRNG and the corresponding b' as

$$\begin{aligned} b' &= (\mathbf{a}')^T\mathbf{s} + \mathbf{e}^T\mathbf{x} + r\lfloor q/2 \rfloor \\ &= \text{LFSR}\,(\text{seed}_{\mathbf{a}'})^T\mathbf{s} + \mathbf{e}^T\mathbf{x} + r\lfloor q/2 \rfloor. \end{aligned}$$

Here $\text{LFSR}(\cdot)$ denotes the output generated by an LFSR.

With LWE parameters chosen as Section IV-A, using a seed of length $l = 256$ is able to reduce the challenge length from 1288 to $256 + 8 = 264$ per one bit of response. The improvement of efficiency becomes more pronounced for generating multiple responses: This is because $\mathbf{a}'_1 \ldots \mathbf{a}'_t$ can be generated sequentially from the l-bit seed, so that only the seed and $b'_1, \ldots, b'_t \in Z_q$ are required to be sent to the PUF side. 100 bits of responses now require only transmitting $256 + 100 \times \log 256 = 1056$ bits for challenges.

C. Countermeasure for Active Attack

The focus of the paper is a PUF secure against passive attacks in which the observed challenges can be used to derive an internal model of the PUF. However, the LWE decryption function is vulnerable to an active attack that supplies arbitrary input challenges. (As we show, this risk also carries into an LFSR-based variant).

The attack is premised on the ability to supply arbitrary challenges (ciphertexts) as inputs to the decryption function. The attack proceeds as follows. The attacker fixes $\mathbf{a}$ and enumerates all possible $b \in \mathbb{Z}_q$ for challenge $\mathbf{c} = (\mathbf{a}, b)$. As b increases from 0 to $q-1$, the response $r = Q(b - \langle \mathbf{a}, \mathbf{b} \rangle)$ changes from $Q(b - \langle \mathbf{a}, \mathbf{s} \rangle) = 0$ to $Q(b + 1 - \langle \mathbf{a}, \mathbf{s} \rangle) = 1$ exactly when b satisfies

$$b - \langle \mathbf{a}, \mathbf{s} \rangle = q/4.$$

We denote this specific value of b as $\hat{b}$. The exact value of $\langle \mathbf{a}, \mathbf{s} \rangle$ can then be extracted by $\langle \mathbf{a}, \mathbf{s} \rangle = \hat{b} - q/4$. By repeating this procedure n times, the attacker is able to set up n linear equations (without errors):

$$\begin{aligned} \langle \mathbf{a}_0, \mathbf{s} \rangle &= \hat{b}_0 - q/4, \\ \langle \mathbf{a}_1, \mathbf{s} \rangle &= \hat{b}_1 - q/4, \\ &\cdots \\ \langle \mathbf{a}_{n-1}, \mathbf{s} \rangle &= \hat{b}_{n-1} - q/4. \end{aligned}$$

Gaussian elimination can then be used to solve for $\mathbf{s}$. The reason the attack succeeds is that attackers are able to fix $\mathbf{a}$ and use it for multiple values of b.

We overcome the risk of such an attack by adopting the technique in [49]: we introduce a self-incrementing counter to embed the counter value into a challenge seed. This makes the attack impossible as the counter restricts the attacker's ability to completely control input challenges to the LWE decryption function. As a result, the attacker cannot enumerate all values of b while keeping $\mathbf{a}$ unchanged. As shown in Figure 1, the concatenation of the challenger-provided seed and the counter value t (i.e. $\text{seed}_{\mathbf{a}'}||t$) is used as the seed for generating $\mathbf{a}$. The counter value is public and is incremented by 1 on each response generation.

V. Experimental Results

In this section, we build a behavior model simulator of the constructed lattice PUF, in which the statistical model of raw SRAM POKs follows from [32], [33] and other digital circuit components are accurately emulated by Python. 1000 lattice PUF instances are virtually manufactured (simulated) and their CRPs are extracted to evaluate (1) statistical properties of the lattice PUF, including uniformity, uniqueness, and reliability with parameters chosen in Section IV, and (2) vulnerability to state-of-the-art ML attacks. In order to quantize the lightweightness, we implement the entire lattice PUF system (except for the raw SRAM cells) on a Spartan 6 FPGA and compare it with prior work.

A. Statistical Analysis

Uniformity of a PUF characterizes unbiasedness, namely, the proportion of '0's and '1's in the output responses. For an ideal PUF f, the proportion needs to be 50%. We adopt the definition of uniformity in [35] based on the average Hamming weight $\text{HW}(f)$ of responses $\mathbf{r}$ to randomly sampled challenges $\mathbf{c}$'s:

$$\text{HW}(f) = \mathbf{E}_{\mathbf{c}}[\text{HW}(\mathbf{r})] = \mathbf{E}_{\mathbf{c}}[\text{HW}(f(\mathbf{c}))].$$

Here $\mathbf{E}_X$ represents expectation over random variable X. Note that $\mathbf{c}$ follows the ciphertext distribution rather than the usual uniform distribution [35]. Figure 4 shows uniformity obtained using 1000 randomly selected challenges. The distribution is centered at 49.98%, the standard deviation is 1.58%.

Uniqueness measures the ability of a PUF to be uniquely distinguished among a set of PUFs. Based on [35], we define this metric to be the average inter-class HD of responses $(\mathbf{r}_i, \mathbf{r}_j)$ under the same challenges $\mathbf{c}$ for a randomly picked PUF pair (f_i, f_j):

$$\text{HD}(f_i, f_j) = \mathbf{E}_{\mathbf{c}}[\text{HD}(\mathbf{r}_i, \mathbf{r}_j)] = \mathbf{E}_{\mathbf{c}}[\text{HD}(f_i(\mathbf{c}), f_j(\mathbf{c}))].$$

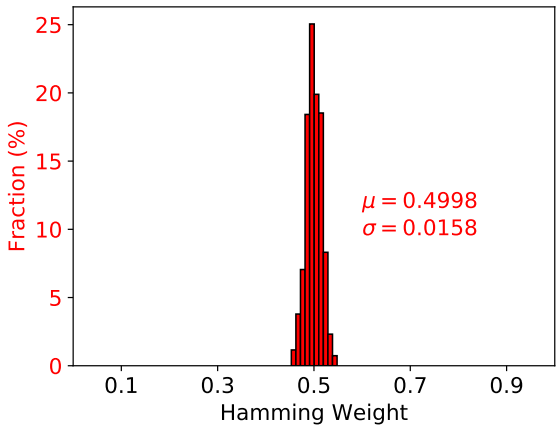

Fig. 4: Uniformity of lattice PUF output.

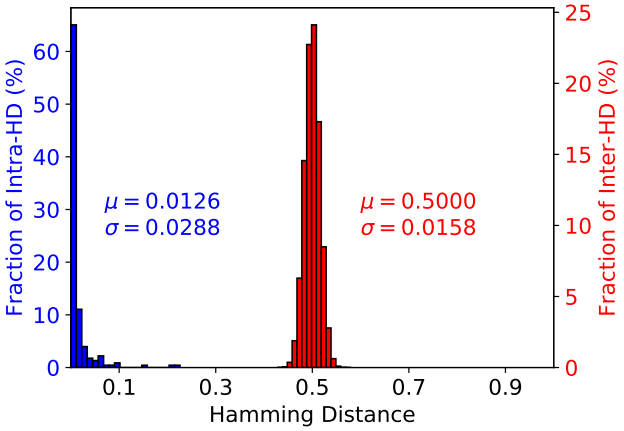

Fig. 5: Uniqueness and reliability of lattice PUF output.

For ideal PUFs, responses under the same challenges are orthogonal, namely, $\mathrm{HD}(f_i, f_j)$'s are close to 50%. Uniqueness is also evaluated under the ciphertext distribution.

Uniqueness is shown in Figure 5, evaluated for 1000 PUF instances. The lattice PUF achieves near-optimal uniqueness: inter-class HD is centered at 50.00%, its standard deviation is 1.58%.

Reliability of a PUF f is characterized by the average BER of outputs with respect to their enrollment values:

$$\mathrm{BER} = \mathbf{E}_{f'}[\mathrm{HD}(f, f')] = \mathbf{E}_{f',\mathbf{c}}[\mathrm{HD}(f(\mathbf{c}), f'(\mathbf{c}))].$$

As discussed in Section IV, the overall BER of the lattice PUF is due to two components: the failure rate of key reconstruction and LWE decryption error rate. Intra-class HD in Figure 5 reflects the result of decryption errors by assuming a perfect key reconstruction.

B. Empirical ML Resistance

While the ultimate promise of lattice PUF is due to its theoretically-supported reliance on hard computational problems, testing its resilience to empirical ML attacks is important and provides additional confidence about the lattice PUF. We

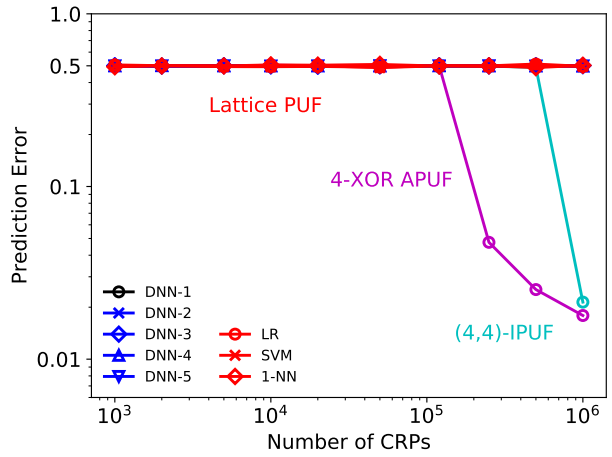

Fig. 6: ML attacks: Lattice PUF remains resistant to all attacks (DNNs, LR, SVM,1-NN). DNN ultimately succeeds in modeling two other strong PUFs.

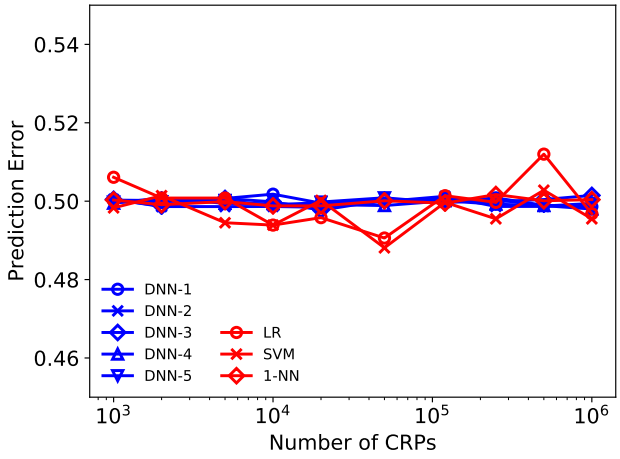

Fig. 7: Lattice PUF is resistant to both traditional ML attacks and DNNs.

evaluate the vulnerability of lattice PUF to a range of traditional (i.e., not based on deep learning) ML attack methods, including SVM, LR, and single-layer NN (1-NN), as well as a number of DNNs. We use the Python package scikit-learn [40] to implement SVM and LR. The SVM uses a nonlinear kernel with radial basis functions (RBF). The 1-NN model uses only one hidden feed-forward layer composed of 100 neurons, with the rectified linear unit (ReLU) as the activation function. Training of 1-NNs and subsequent DNNs are implemented using Keras [12] with TensorFlow [1] backend.

DNNs represent a more powerful class of binary classifiers. DNNs contain multiple hidden layers that produce superior modeling expressiveness compared to 1-NNs [18]. Importantly, DNNs have been recently shown to be very effective in attacking known strong PUFs, including XOR APUFs and IPUFs [43]. Our baseline DNN experiment (DNN-1) is based on the same network parameters as in [43]. The network has

TABLE I: Various configuration for DNN attacks.

Setup	Hidden Layers	Neurons per Layer	Challenge Distribution	Input Format	Prediction Error
DNN-1	4	100	PRNG	Binary	49.86%
DNN-2	4	100	PRNG	Real	49.84%
DNN-3	4	100	Ciphertext	Binary	49.76%
DNN-4	6	100	PRNG	Binary	49.80%
DNN-5	4	200	PRNG	Binary	49.87%

TABLE II: Configuration of error-correcting codes.

Raw BER (%)	**Error-Correcting Code**		**Raw POKs**
	Outer code	Inner code	
1	[236, 128, 14]	N/A	2,360
5	[218, 128, 11]	[3, 1, 1]	6,540
10	[220, 128, 12]	[5, 1, 2]	11,000
15	[244, 128, 15]	[7, 1, 3]	17,080

4 hidden layers, each containing 100 neurons. It uses ReLU as the non-linear operator.

In addition to the baseline configuration, we explored attacks using several other network architectures, hyper-parameters, and numeric representations, as listed in Table I. DNN-2 treats input as 161 integer numbers from 0 to 255, instead of 1288 binary bits. DNN-3 is different from the baseline version in its CRP generation strategy (see more below). DNN-4 and DNN-5 add more hidden layers and more neurons per hidden layer respectively, compared to the baseline DNN-1.

Figure 6 shows the results of the empirical attacks based on the above ML algorithms. The figure shows the prediction error of lattice PUF in response to these attacks with training set size ranging from 1000 to 1 million and test set of size 200K. The Adam optimizer [27] terminates after 200 epochs, and results in a prediction error of 49.86% for the proposed lattice PUF, barely better than a random guess. *The results show that the prediction error of lattice PUF remains flat for all attempted attacks: across the range of attacks and CRP training set sizes, there is no measurable deviation of the error from 0.5.* In contrast, a DNN (with a configuration corresponding to DNN-1) achieves less than 2% prediction error for both 4-XOR APUF and (4, 4)-IPUF.

It is critical that the experiments also demonstrate that lattice PUF design that utilizes the distributional relaxation of space-efficient LWE (described in Section IV-B) shows the same empirical resistance to ML attacks as the design not based on such a relaxation. In Table 1, all design/attack combinations except for DNN-3 are based on the compact (relaxation-based) design in which CRPs are generated via a PRNG. Finally, we also provide an expanded view of the results of ML attacks on lattice PUF in Figure 7 by zooming in Figure 6. While run-to-run variations of the training optimizer are observable, the prediction error remains close to 50%.

C. Hardware Implementation Results

We now present the details of lattice PUF implementation and analyze its hardware efficiency. The entire design, except for the raw (uninitialized) SRAM cells, was synthesized, configured, and tested on a Xilinx Spartan-6 FPGA (XC6SLX45), a low-end FPGA in 45nm technology.

TABLE III: (a) Area consumption and (b) runtime of our reference lattice PUF implementation on Xilinx Spartan-6 FPGA.

(a)

Module	**Size [slices]**
LFSR	27
LWEDec	2
Controller	16
Total	45

(b)

Step	**Time [μs]**
Seed $\text{seed}_{\mathbf{a}'}||t$ load for LFSR	8
1-bit decryption from LWEDec	44
Total @ 33 MHz	52

TABLE IV: Hardware implementation costs of strong PUFs.

Design	**Platform**	**PUF Logic [Slices]**
POK+AES [7]	Spartan 6	80
Controlled PUF [17]	Spartan 6	127
CFE-based PUF [19], [21]	Zynq-7000	9,825
Lattice PUF	Spartan 6	45

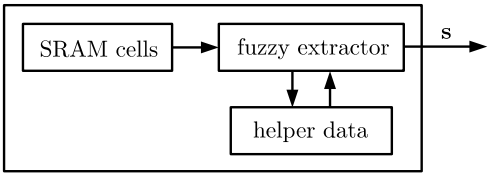

Fig. 8: POK uses an FE to ensure stability of the secret seed.

Regarding the FE design, we adopt the homogeneous error assumption, i.e., all cells have the same BER [8]. Prior work shows that intrinsic BERs of the various POKs range from 0.1% [23] to 15% [33]. We study the POK and FE designs under four levels of raw BER: 1%, 5%, 10%, and 15% to explore design costs, and choose 5% as the raw SRAM BER to benchmark our final implementation. Here the goal of FE design is to ensure a key reconstruction of 1280 bits with targeted failure rate at 10^{-6}. As mentioned in Section IV, with such nearly-perfect secret key reconstruction, the overall output BER of the lattice PUF can reach the decryption error rate analyzed above. We use concatenated error-correcting codes, with a repetition code as the inner code, and a shortened BCH code as the outer code. Concatenated codes are typically more efficient than single codes in terms of code length and hardware cost [8]. Block diagram of the FE design is shown in Figure 8. Table II and V list the configuration and hardware costs of error-correcting codes used at different BER levels respectively. At the raw BER of 5%, $6.5K$ cells are needed to construct the secret $\mathbf{s}$ of length 1280 bits at the target failure

TABLE V: Hardware utilization in FE design on Spartan 6 FPGA.

Raw BER (%)	Outer Code			Inner Code			Total		
	Reg	LUT	Slice	Reg	LUT	Slice	Reg	LUT	Slice
1	905	893	276	0	0	0	905	893	276
5	730	688	232	0	1	1	730	689	233
10	785	740	243	0	3	2	785	743	245
15	973	913	326	0	7	3	973	920	329

rate 10^{-6}. The FE design of the lattice PUF requires 233 slices. This portion of cost applies to all other strong PUF candidates (AES PUF or other controlled PUF). This is also cheaper than linear solver block used in the CFE-based strong PUF [19], [21] for key reconstruction, which requires $65,700$ LUTs and $16,425$ slices.

Regarding the PUF logic, the total size of the lattice PUF (without FE) for the Spartan-6 platform is 45 slices, most of which is taken up by the LFSR and the controller. Table IIIa shows the breakdown of resources needed to realize the various modules. The core block implementing the LWE decryption function (LWEDec) includes an 8-bit MAC and a quantization block, as shown in Figure 2. The 256-bit LFSR is implemented using RAM-based shift registers. The total latency (at 33.3MHz clock) to generate a 1-bit PUF response is 47μs, and the total time to generate a 100-bit PUF response is, approximately, $8\mu s + 100 \times 44\mu s \approx 4.4ms$ since seed loading is only executed once. Table IIIb lists the latency of each step of response generation.

We compare the implementation cost of the lattice PUF logic against established strong PUF designs [7], [17], [21] in Table IV. The original strong PUF based on AES [7] is implemented as an ASIC. Here, we adopt [13] as an FPGA alternative to estimate the implementation cost of AES. Notice that [7] uses no error correction since it guarantees reliability via dark bit masking. Similarly, the FPGA implementation of SHA-3 [22] is adopted to estimate the cost of a hash function for the controlled PUF [17]. The FPGA utilization result of the strong PUF based on the computational FE (CFE) is presented via the number of LUTs in [21]. We estimate the corresponding slice count using [48]. Compared to PUFs based on AES [13] and SHA [22], our advantages in area are minor. However, compared to [19], [21], which is another PUF based on LWE, and which therefore provides similar theoretical guarantees, our savings in area are significant.

VI. Conclusion

In this paper, we described a new strong physical unclonable function (PUF) that is provably secure against machine learning (ML) attacks with both classical and quantum computers. The security is derived from cryptographic hardness of learning decryption functions of semantically secure public-key cryptosystems within the probably approximately correct framework. The proposed PUF compactly realizes the decryption function of the learning-with-errors (LWE) public-key cryptosystem as the core block. We implemented a lattice PUF on a Spartan 6 FPGA. The design realizes a challenge-response pair space of size 2^{136}, requires 1280 physically obfuscated key bits, and guarantees 128-bit ML resistance. The PUF shows excellent uniformity, uniqueness, and reliability.

Acknowledgments

We thank Dr. Aydin Aysu for his insightful advice on idea presentation, assistance with FPGA implementation of repetition code, and comments that greatly improved the manuscript.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *Theory of Cryptography Conference*, pages 474–495. Springer, 2009.

[3] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[4] D. Apon, C. Cho, K. Eldefrawy, and J. Katz. Efficient, reusable fuzzy extractors from lwe. In *International Conference on Cyber Security Cryptography and Machine Learning*, pages 1–18. Springer, 2017.

[5] G. T. Becker. The gap between promise and reality: On the insecurity of xor arbiter pufs. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 535–555. Springer, 2015.

[6] D. J. Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.

[7] M. Bhargava and K. Mai. An efficient reliable puf-based cryptographic key generator in 65nm cmos. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 70. European Design and Automation Association, 2014.

[8] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls. Efficient helper data key extractor on fpgas. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 181–197. Springer, 2008.

[9] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013.

[10] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, and U. Rührmair. The bistable ring puf: A new architecture for strong physical unclonable functions. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 134–141. IEEE, 2011.

[11] Y. Chen and P. Q. Nguyen. Bkz 2.0: Better lattice security estimates. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2011.

[12] F. Chollet et al. Keras, 2015.

[13] J. Chu and M. Benaissa. Low area memory-free fpga implementation of the aes algorithm. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 623–626. IEEE, 2012.

[14] B. Fuller, X. Meng, and L. Reyzin. Computational fuzzy extractors. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 174–193. Springer, 2013.

[15] S. D. Galbraith. Space-efficient variants of cryptosystems based on learning with errors. *url: https://www. math. auckland. ac. nz/~ sgal018/compact-LWE. pdf*, 2013.

[16] F. Ganji, S. Tajik, F. Fäßler, and J.-P. Seifert. Strong machine learning attack against pufs with no mathematical model. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 391–411. Springer, 2016.

[17] B. Gassend, M. V. Dijk, D. Clarke, E. Torlak, S. Devadas, and P. Tuyls. Controlled physical random functions and applications. *ACM Transactions on Information and System Security (TISSEC)*, 10(4):3, 2008.

[18] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[19] C. Herder, L. Ren, M. van Dijk, M.-D. Yu, and S. Devadas. Trapdoor computational fuzzy extractors and stateless cryptographically-secure physical unclonable functions. *IEEE Transactions on Dependable and Secure Computing*, 14(1):65–82, 2017.

[20] N. Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against ntru. In *Annual International Cryptology Conference*, pages 150–169. Springer, 2007.

[21] C. Jin, C. Herder, L. Ren, P. H. Nguyen, B. Fuller, S. Devadas, and M. van Dijk. Fpga implementation of a cryptographically-secure puf based on learning parity with noise. *Cryptography*, 1(3):23, 2017.

[22] J.-P. Kaps, P. Yalla, K. K. Surapathi, B. Habib, S. Vadlamudi, S. Gurung, and J. Pham. Lightweight implementations of sha-3 candidates on fpgas. In *International Conference on Cryptology in India*, pages 270–289. Springer, 2011.

[23] B. Karpinskyy, Y. Lee, Y. Choi, Y. Kim, M. Noh, and S. Lee. 8.7 physically unclonable function for secure key generation with a key error rate of 2e-38 in 45nm smart-card chips. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*, pages 158–160. IEEE, 2016.

[24] J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[25] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[26] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 372–381. ACM, 1993.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] A. R. Klivans and A. A. Sherstov. Cryptographic hardness for learning intersections of halfspaces. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 553–562. IEEE, 2006.

[29] R. Kumar and W. Burleson. On design of a highly secure puf based on non-linear current mirrors. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 38–43. IEEE, 2014.

[30] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205, 2005.

[31] R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *Cryptographers' Track at the RSA Conference*, pages 319–339. Springer, 2011.

[32] R. Maes. An accurate probabilistic reliability model for silicon pufs. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 73–89. Springer, 2013.

[33] R. Maes, P. Tuyls, and I. Verbauwhede. A soft decision helper data algorithm for sram pufs. In *2009 IEEE international symposium on information theory*, pages 2101–2105. IEEE, 2009.

[34] R. Maes, A. Van Herrewege, and I. Verbauwhede. Pufky: A fully functional puf-based cryptographic key generator. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 302–319. Springer, 2012.

[35] A. Maiti, V. Gunreddy, and P. Schaumont. A systematic method to evaluate and compare the performance of physical unclonable functions. In *Embedded systems design with FPGAs*, pages 245–267. Springer, 2013.

[36] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Lightweight secure pufs. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 670–673. IEEE Press, 2008.

[37] D. Micciancio and O. Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, 2009.

[38] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[39] P. H. Nguyen, D. P. Sahoo, C. Jin, K. Mahmood, U. Rührmair, and M. van Dijk. The interpose puf: Secure puf design against state-of-the-art machine learning attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 243–290, 2019.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[42] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 237–249. ACM, 2010.

[43] P. Santikellur, A. Bhattacharyay, and R. S. Chakraborty. Deep learning based model building attacks on arbiter PUF compositions. *IACR Cryptology ePrint Archive*, 2019:566, 2019.

[44] D. Schuster and R. Hesselbarth. Evaluation of bistable ring pufs using single layer neural networks. In *International Conference on Trust and Trustworthy Computing*, pages 101–109. Springer, 2014.

[45] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, pages 9–14. ACM, 2007.

[46] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu. Machine learning resistant strong puf: Possible or a pipe dream? In *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*, pages 19–24. IEEE, 2016.

[47] X. Xi, H. Zhuang, N. Sun, and M. Orshansky. Strong subthreshold current array puf with 2 65 challenge-response pairs resilient to machine learning attacks in 130nm cmos. In *VLSI Circuits, 2017 Symposium on*, pages C268–C269. IEEE, 2017.

[48] Xilinx. *Zynq-7000 SoC Data Sheet: Overview*, 7 2018. v1.11.1.

[49] M.-D. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, and I. Verbauwhede. A lockdown technique to prevent machine learning on pufs for lightweight authentication. *IEEE Transactions on Multi-Scale Computing Systems*, 2(3):146–159, 2016.