

Beasley: Efficient Zero-Knowledge Proofs for Lattice-based Round-Optimal Oblivious Pseudorandom Functions

Baiyu Li*
baiyuli@google.com

Ting-Yuan Wang[†]
twang151@usc.edu

Jiapeng Zhang[†]
jiapengz@usc.edu

Abstract

Verifiable oblivious pseudorandom functions (VOPRFs) enable a client to evaluate a pseudorandom function on a private input under a server-held key while verifying that the server evaluated the function honestly using its committed key. Existing practical VOPRFs rely mainly on assumptions that are vulnerable to quantum attacks. Prior lattice-based constructions achieving both round optimality and malicious security have remained theoretical proposals without concrete implementations, largely because proving the correct evaluation of the underlying lattice-based PRF in zero-knowledge is prohibitively expensive.

We present Beasley, the first concrete prototype implementation of a round-optimal, maliciously secure, lattice-based VOPRF from standard lattice assumptions. Beasley builds upon the framework of Albrecht et al. (PKC 2021) and LeOPaRd (CRYPTO 2026), introducing two core optimizations: First, we generalize the Boneh-Lewi-Montgomery-Ragunathan (BLMR13) pseudorandom function where each step processes w input bits simultaneously; instantiated with $w = 4$ in our implementation, this reduces evaluation depth and running time by a factor of 4 without impact on security; Second, we replace LaBRADOR-style proofs with a ring-switching sumcheck protocol and develop an efficient arithmetization of the OPRF computation. This design allows us to use a relatively large ring dimension of 512, simultaneously delivering fast ring arithmetic and compact keys with efficient exact range proofs without security-degrading slack.

Benchmarked on a commodity consumer laptop using a single core with AVX2, Beasley generates the full client request and NIZK proof in 520 ms, with a server verification time of 15.3 ms, a peak RAM footprint of 113.7 MB, and a client communication size of 109 KB. The NIZK proof itself is 75.4 KB. Our results demonstrate that maliciously secure, lattice-based VOPRFs can achieve practical performance for real-world applications.

1 Introduction

Verifiable Oblivious Pseudorandom Functions (VOPRFs) are essential cryptographic primitives with broad applications in privacy-preserving systems. At a high level, a VOPRF is a two-party protocol that allows a client holding a private input x to learn the output of a pseudorandom function $F_{\mathbf{k}}(x)$ under a master secret key $\mathbf{k}$ held by a server. The protocol guarantees *obliviousness*: the server learns nothing about the client’s input x or the resulting output value $F_{\mathbf{k}}(x)$. At the same time, the *verifiability* property assures the client that the server evaluated the function F correctly using $\mathbf{k}$. Together, these properties ensure that neither a malicious server nor a malicious client can deviate from the protocol or manipulate the interaction to extract unauthorized information.

Due to their robust privacy and integrity guarantees, VOPRFs serve as foundational building blocks in modern security infrastructure [TCR⁺22, FLÖ25, DGSW24]. Widely deployed examples include *Privacy Pass* [DGSW24], which enables anonymous authorization for web clients to bypass repeated CAPTCHA

*Google. Author names are listed in alphabetical order.

[†]Thomas Lord Department of Computer Science, University of Southern California. Research supported by a Google PSS Faculty Award.

challenges without tracking browsing habits across sites, and *Private Set Intersection (PSI)*, which allows mutually distrusting organizations to discover common records without exposing unmatched data. VOPRFs are also central to modern user authentication and notably, they form the backbone of the *OPAQUE* protocol [JKK14], which is an asymmetric Password-Authenticated Key Exchange (aPAKE) standard that protects user credentials against offline dictionary attacks even if the server database is completely compromised. Furthermore, VOPRFs are increasingly deployed in anonymous credential systems, privacy-preserving telemetry and rate-limiting services, enabling service providers to authenticate or meter users without tracking their long-term identities.

However, the rapid progress in quantum computing poses an existential threat to these deployed systems. Standardized VOPRF protocols (e.g., RFC 9497 [DGSW24]) rely almost exclusively on classical discrete logarithm and Diffie–Hellman assumptions in prime-order elliptic-curve groups, which are vulnerable to polynomial-time quantum attacks via Shor’s algorithm. While post-quantum cryptography has seen significant progress in deploying signatures and key encapsulation mechanisms (KEMs), designing post-quantum VOPRFs that are simultaneously *round-optimal*, *maliciously secure*, and *concretely efficient* remains a notoriously difficult task. Given the ubiquitous role of VOPRFs in privacy infrastructure, constructing practical, quantum-resistant VOPRFs is an urgent practical necessity.

1.1 Challenges of Implementing PQ-VOPRFs

While lattice-based cryptography provides a promising foundation for post-quantum security, realizing practical, round-optimal VOPRFs from standard lattice assumptions remains difficult. To the best of our knowledge, prior to this work, no concrete, fully malicious-secure lattice-based VOPRF protocol had been implemented. To understand the core obstacles, we recall the blueprint for round-optimal lattice-based VOPRFs initiated by Albrecht, Davidson, Deo, and Smart (ADDS21) [ADDS21] and subsequently refined by LeOPaRd [ESTX26].

The ADDS21 / LeOPaRd Blueprint. The framework of [ADDS21, ESTX26] represents a lattice analog of the classical 2HashDH construction [JKK14]. The underlying PRF has the form $\lfloor p/q \cdot (\mathbf{H}(x) \cdot \mathbf{k} \pmod{q}) \rfloor$, where $\mathbf{k}$ is the server’s secret key, $x \in \{0,1\}^n$ is the client’s input, and $\mathbf{H}$ is a public *message mapping* function. This template captures several Learning With Rounding (LWR) based PRF constructions such as [BLMR13, BP14, BBL⁺14]. The protocol executes in a single round trip:

1. **Setup:** The server samples a secret key $\mathbf{k}$, and publishes a random matrix $\mathbf{A}$ together with a public key $\mathbf{pk} = \mathbf{A} \cdot \mathbf{k} + \mathbf{e} \pmod{q}$.
2. **Client Request:** For any input $x \in \{0,1\}^n$, the client samples short noise $\mathbf{s}$ and masks $\mathbf{H}(x)$ as

$$\mathbf{c}_x = \mathbf{s} \cdot \mathbf{A} + \mathbf{H}(x) \pmod{q}.$$

It also proves in zero-knowledge that $\mathbf{c}_x$ is well-formed.

3. **Server Evaluation:** The server verifies π_c and evaluates the request blindly by computing

$$d_x = \mathbf{c}_x \cdot \mathbf{k} + e_2 \pmod{q}$$

using its secret key $\mathbf{k}$, where e_w is a freshly sampled noise term to hide dependency of $\mathbf{k}$. The server attaches an NIZK proof π_s certifying that d_x was evaluated under the committed key $\mathbf{k}$.

4. **Client Finalize:** The client verifies π_s , and unblinds mask by computing

$$z = d_x - \mathbf{s} \cdot \mathbf{pk} = \mathbf{H}(x) \cdot \mathbf{k} + (e_2 - \mathbf{s} \cdot \mathbf{e}) \pmod{q}.$$

The PRF output is then $\lfloor p/q \cdot z \rfloor$.

As long as the accumulated noise $e_2 - \mathbf{s} \cdot \mathbf{e}$ is relatively small, the rounding operation can eliminate the noise and yields the correct PRF output. Client and (in particular) server security can be achieved by choosing suitable noise distributions, and as shown in [ESTX26], we can tweak this construction to use quasi-optimal noise parameters.

The challenge. While the protocol is conceptually clean and elegant, ensuring security against malicious adversaries requires both parties to prove the correctness of their messages in zero knowledge. Take the client request as an example. The client must prove knowledge of x and short $\mathbf{s}$ such that $\mathbf{c}_x = \mathbf{s} \cdot \mathbf{A} + \mathbf{H}(x) \pmod{q}$. Without this proof, a malicious client could carefully choose an element that is not an image of $\mathbf{H}$ to recover the server’s secret key $\mathbf{k}$ [ADDS21]. Popular choices of $\mathbf{H}$ include lattice-based PRFs, such as the Boneh-Lewi-Montgomery-Raghuathan PRF [BLMR13] or the Banerjee-Peikert PRF [BP14]; or directly use a random oracle. The main challenge is to efficiently generate a succinct NIZK proof of $\mathbf{H}(x)$. Prior works struggled to instantiate these proofs efficiently:

- ADDS21 [ADDS21] used standard SNARKs with field simulation, resulting in prohibitively large proof (> 128 GB).
- Subsequent works (AG24 [AG24] and LeOPaRd [ESTX26]) proposed using LaBRADOR [BS23] to reduce proof sizes below 100 KB. However, LaBRADOR is tailored for approximate ℓ_2 norm bounds and can be expensive to enforce exact ℓ_∞ bounds. Furthermore, it is optimized for small ring dimensions (e.g. $N = 64$ or $N = 128$) which is not ideal for computing $\mathbf{H}(x)$, and it incurs non-satisfactory proof generation time (estimated at several minutes in LeOPaRd [ESTX26]) and high verification times.

Consequently, a practical, implemented post-quantum VOPRF remained an elusive goal.

1.2 Our Contributions

In this paper, we resolve these challenges and design efficient proof systems for the BLMR13 PRF and subsequently the LeOPaRd VOPRF [ESTX26]. We present **Beasley**, the first fully implemented, round-optimal, maliciously secure lattice-based VOPRF. Beasley demonstrates that lattice VOPRFs can achieve sub-second client latency, millisecond-level verification, and compact proof sizes on commodity hardware. Although we follow the LeOPaRd protocol design, we remark that achieving a concretely efficient implementation remains non-trivial and requires overcoming a number of obstacles. Our technical contributions span two key areas:

1. **Generalized BLMR13 PRF:** We generalize the Boneh-Lewi-Montgomery-Raghuathan PRF [BLMR13] to improve the evaluation time of $\mathbf{H}(x)$ and hence reduce its NIZK proof complexity. Rather than processing inputs bit-by-bit, we group the input into $k = n/w$ blocks of w bits each. We then define $\mathbf{H}(x)$ as a product of k matrices where each matrix is selected from 2^w public matrices $\mathbf{A}_z$ based on the bits in the corresponding input block. The public matrices $\mathbf{A}_z$ are generated from short seeds using a PRG to keep communication overhead small. We set $w = 4$ in our implementation, which reduces the number of sequential matrix multiplications from 128 to 32, yielding an immediate $4\times$ reduction in both the PRF evaluation and proof generation times, while preserving security guarantees.
2. **Ring-Switching Based Proof System:** Instead of relying on LaBRADOR as suggested in [ESTX26], we design new proof systems based on the ring-switching technique of Huang, Mao, and Zhang [HMZ26]. Ring switching translates relations over high degree cyclotomic ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ into polynomial constraints over an extension field $\mathbb{F}_{q^d}$ of small d . This yields three advantages:
 - *Exact range proofs:* Our proof systems support exact range proofs while keeping the overall cost low, allowing us to use optimal parameters without impact on the security proof.
 - *Decouple $\mathcal{R}_q$ from the proof systems:* We follow the sumcheck style proof whose size is determined by the multilinear witness size rather than the degree of $\mathcal{R}_q$, so we can set the ring degree to $N = 512$ with a smaller module rank $m = 5$. This substantially accelerates ring arithmetic compared to the parameters in LeOPaRd [ESTX26], where they picked $N = 64$ and $m = 40$ to fit the LaBRADOR.
 - *Concrete efficiency:* The client latency drops from minutes to 520 ms, while the verifier time is reduced to only 15.3 ms (a $> 60\times$ speedup over LaBRADOR verification). In particular, several papers [HMZ26, NOZ26] have shown that avoiding the Johnson–Lindenstrauss projection in LaBRADOR, yields ring switching an asymptotic $\Theta(\lambda)$ -factor improvement in verification time.

Protocol	Hardness Assumption	Security Model	Rounds	Client Time	Verify Time	Client Comm.
ADDS21 [ADDS21]	Ring-LWE / Ring-SIS	Malicious	2	No estimation	No estimation	> 128 GB
AG24 [AG24]	MLWE / MSIS	Malicious	2	Minutes (est.)	> 1 s (est.)	$\approx$ 86 KB (est.)
LeOPaRd [ESTX26]	MLWE / MSIS	Malicious	2	Minutes (est.)	> 1 s (est.)	$\approx$ 75 KB (est.)
LEAP [HKL ⁺ 25]	LWR / OT	Semi-Honest only	3	11 ms	N/A	23 KB
POOL [DDTT25]	LWR / OT	Semi-Honest only	2	2 ms	N/A	11.9 KB
Gold OPRF [YBH ⁺ 25]	DSPRS (Non-standard)	Malicious	2	510 ms	507 ms	970 KB
Beasley (Ours)	MLWE / MSIS	Malicious	2	520 ms	15.3 ms	109 KB

Table 1: Comparison of Post-Quantum OPRF / VOPRF Protocols. The last column measure the size of client request, which for malicious setting is a sum of raw request (w/o NIZK) and the NIZK proof π_c . For AG24, LeOPaRd, and Beasley, we consider parameters providing 128 bits security for up to 2^{32} queries.

3. **Prototype Implementation and Benchmarks:** We implement Beasley in Rust using AVX2 SIMD vectorization and evaluate it on a commodity Intel Core i5-13600K CPU. As summarized in Table 3, Beasley generates the complete client request in 520 ms with 113.7 MB of peak RAM and a proof size of 75.4 KB.

1.3 Related Works

Post-quantum OPRFs have received substantial attention recently. See [AAS25] for a slightly dated survey. Below, we categorize existing approaches and compare their properties (summarized in Table 1).

Lattice-Based VOPRFs. The earliest round-optimal lattice VOPRF by Albrecht et al. (ADDS21) [ADDS21] established the blind LWR evaluation paradigm. However, they relied on generic field-simulation SNARKs which resulted in impractical communication cost (> 128 GB). Albrecht and Gur (AG24) [AG24] and LeOPaRd [ESTX26] achieved succinct proofs by applying LaBRADOR [BS23]. LeOPaRd further improved the ADDS21 blueprint with optimal noise distributions that achieve full malicious security with minimal server cost. However, LaBRADOR is optimized for small ring dimensions ($N = 64$ or $N = 128$) and approximate ℓ_2 -norm bounds, and hence it requires larger VOPRF parameters to compensate the norm-slack. In addition, its estimated performance is not yet satisfactory.

Oblivious Transfer (OT) Based OPRFs. Another line of research constructs post-quantum OPRFs from Oblivious Transfer, notably LEAP [HKL⁺25] and POOL [DDTT25]. These constructions achieve ultra-low client evaluation latency (< 2 ms) and small communication (16–48 KB). However, they achieve *only semi-honest security*. As highlighted by the authors of POOL [DDTT25], upgrading OT-based OPRFs to full malicious security requires the client to attest to the correct evaluation of $H(x)$ on private input x . Our approach differs from such direction in that we instantiate H with the message mapping function in the BLMR13 PRF, and we provide an NIZK proof for its computation.

VOLE-in-the-Head and Power-Residue PRFs. A third direction utilizes Vector Oblivious Linear Evaluation (VOLE) and power-residue PRFs, exemplified by Gold OPRF [YBH⁺25]. Gold OPRF achieves competitive client latency and single-round online interaction. However, Gold OPRF suffers from two drawbacks: first, its proof size is relatively large (970 KB); second, and more importantly, its security relies on the *Decisional Short Power Residue Sum (DSPRS)* problem, which is relatively new assumption that has not undergone extensive cryptanalysis. In contrast, Beasley relies exclusively on standard, well-studied MLWE and MSIS assumptions.

Use of Generative AI. AI tools were used for editorial assistance, primarily to improve the grammar and clarity of author-written text, and to assist with parts of the implementation. All AI-assisted text was

reviewed and revised by the authors. All AI-assisted code was manually inspected, tested, and verified by the authors before being used in our implementation or experiments. The authors take full responsibility for the correctness, originality, and integrity of all content, implementations, and experimental results.

2 Preliminaries

Notation. Let q be a prime. We write $\mathbb{Z}_q := \{0, 1, \dots, q-1\}$ for the ring of integers modulo q , and $\mathbb{F}_{q^d}$ for the finite field of order q^d . For any positive integer n , we define $[n] := \{0, 1, \dots, n-1\}$. Let λ denote the computational security parameter. For a distribution $\mathcal{X}$ (resp. finite set $\mathcal{X}$), $x \leftarrow \mathcal{X}$ indicates that x is sampled according to $\mathcal{X}$ (resp. chosen uniformly at random from $\mathcal{X}$). We write $\text{negl}(\lambda)$ to denote an unspecified negligible function. We use bold lowercase letters (e.g., $\mathbf{a}, \mathbf{b}$) for vectors and bold uppercase letters (e.g., $\mathbf{A}, \mathbf{B}$) for matrices over a given ring. By default, vectors are column vectors, and $\mathbf{a}^\top$ denotes the transpose of $\mathbf{a}$. For an element $a \in \mathcal{R}_q$, $\|a\|_\infty$ denotes the maximum absolute value of its coefficients using the canonical representations in $[-q/2, q/2) \cap \mathbb{Z}$, and for a vector $\mathbf{a} = (a_1, \dots, a_m)^\top \in \mathcal{R}_q^m$, $\|\mathbf{a}\|_\infty := \max_i \|a_i\|_\infty$. For $x \in \mathbb{R}$, $\lfloor x \rfloor$ denotes rounding to the nearest integer. For moduli $0 < p < q$ and $z \in \mathbb{Z}_q$, we define $\lfloor z \rfloor_p := \lfloor (p/q) \cdot z \rfloor \bmod p$, extended coefficient-wise to polynomials and vectors.

Cyclotomic Rings and Gadget Decompositions. Let N be a power of two, and let q be a prime modulus. We define the $2N$ -th cyclotomic ring and its quotient ring modulo q as:

$$\mathcal{R} = \mathbb{Z}[X]/(X^N + 1), \quad \mathcal{R}_q = \mathcal{R}/q\mathcal{R} \cong \mathbb{Z}_q[X]/(X^N + 1).$$

Let $b \geq 2$ be an integer gadget base, and let $\ell = \lceil \log_b q \rceil$. We define the base- b gadget vector as

$$\mathbf{g} = (1, b, b^2, \dots, b^{\ell-1}) \in \mathcal{R}_q^\ell$$

For any element $z \in \mathcal{R}_q$, let $\mathbf{g}^{-1}(z) \in \mathcal{R}^\ell$ denote its deterministic coefficient-wise base- b gadget decomposition such that every coefficient of $\mathbf{g}^{-1}(z)$ lies in $\{0, 1, \dots, b-1\}$, and $\mathbf{g}^\top \cdot \mathbf{g}^{-1}(z) = z \pmod{q}$. We generalize this to module rank $m \geq 1$ by defining the block-diagonal gadget matrix $\mathbf{G}_m = \mathbf{I}_m \otimes \mathbf{g}^\top \in \mathcal{R}_q^{m \times m\ell}$. The corresponding component-wise gadget decomposition $\mathbf{G}_m^{-1} : \mathcal{R}_q^m \rightarrow \mathcal{R}_q^{m\ell}$ satisfies for all $\mathbf{z} \in \mathcal{R}_q^m$:

$$\mathbf{G}_m^{-1}(\mathbf{z}) \in \{0, \dots, b-1\}^{Nm\ell} \quad \wedge \quad \mathbf{G}_m \mathbf{G}_m^{-1}(\mathbf{z}) = \mathbf{z} \pmod{q}.$$

When the context is clear, we simply write $\mathbf{G}$ and $\mathbf{G}^{-1}$.

Hard Lattice Problems. The security of our VOPRF construction relies on the hardness of standard Module-SIS and Module-LWE problems over the cyclotomic ring $\mathcal{R}_q$.

Definition 2.1 ($\text{MSIS}_{m,k,\mathcal{R}_q,\beta}$). Let $m, k > 0$ be integers, and let $\beta > 0$. The Module Short Integer Solution ($\text{MSIS}_{m,k,\mathcal{R}_q,\beta}$) problem asks, given a uniformly random matrix $\mathbf{A} \leftarrow \mathcal{R}_q^{m \times k}$, to find a non-zero vector $\mathbf{z} \in \mathcal{R}_q^k \setminus \{\mathbf{0}\}$ such that

$$\mathbf{A} \cdot \mathbf{z} = \mathbf{0} \pmod{q} \quad \text{and} \quad 0 < \|\mathbf{z}\|_2 \leq \beta.$$

The $\text{MSIS}_{m,k,\mathcal{R}_q,\beta}$ assumption states that for any efficient adversary $\mathcal{A}$, $\Pr\{\mathcal{A}(\mathbf{A}) \text{ solves MSIS}\} \leq \text{negl}(\lambda)$.

Definition 2.2 ($\text{MLWE}_{m,k,\mathcal{R}_q,\chi_s,\chi_e}$). Let $m, k > 0$ be integers, and let χ_s, χ_e be error distributions over $\mathcal{R}$. The decisional Module Learning With Errors ($\text{MLWE}_{m,k,\mathcal{R}_q,\chi_s,\chi_e}$) assumption states that the distribution

$$\{(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \pmod{q}) \mid \mathbf{A} \leftarrow \mathcal{R}_q^{k \times m}, \mathbf{s} \leftarrow \chi_s^m, \mathbf{e} \leftarrow \chi_e^k\}$$

is computationally indistinguishable from the uniform distribution over $\mathcal{R}_q^{k \times m} \times \mathcal{R}_q^k$. We sometimes omit the number of samples k and write $\text{MLWE}_{m,\mathcal{R}_q,\chi_s,\chi_e}$ when k polynomially bounded in mN .

BLMR PRF. We now review the lattice-based pseudorandom function by Boneh, Lewi, Montgomery, and Raghunathan [BLMR13], adopted to the module lattice setting.

Let $0 < p < q$ be integer moduli. The public parameters consist of two uniform and independent binary matrices $\mathbf{A}_0, \mathbf{A}_1 \in \{0, 1\}^{m\ell \times m\ell}$ conditioned on being both full rank over $\mathcal{R}_q$. The secret key $\mathbf{k} \leftarrow \mathcal{R}_q^{m\ell}$ is sampled uniformly at random from $\mathcal{R}_q^{m\ell}$. For an input $x = (x_0, \dots, x_{n-1}) \in \{0, 1\}^n$, we first compute

$$\mathbf{H}^{\text{blmr}}(x) = \prod_{i=0}^{n-1} \mathbf{A}_{x_i}.$$

Since $x_i \in \{0, 1\}$, the selection of the public matrix at the i -th level can equivalently be written as $\mathbf{A}_{x_i} = (1 - x_i) \cdot \mathbf{A}_0 + x_i \cdot \mathbf{A}_1$. The BLMR PRF is then defined as

$$F_{\mathbf{k}}^{\text{blmr}}(x) = \lfloor \mathbf{k}^\top \cdot \mathbf{H}^{\text{blmr}}(x) \rfloor_p \in \mathcal{R}_p^{m\ell},$$

where rounding is applied coefficient-wise to the nearest integer and the result is reduced modulo p .

The BLMR PRF relies on a variant of MLWE problem called *Non-uniform Learning With Errors* (NLWE) in which the columns of the $\mathbf{A}$ matrix are sampled from a low-norm distribution over $\mathcal{R}_q^{m\ell}$. Under suitable parameters, the NLWE problem can be reduced from standard MLWE [BLMR13].

BP14 PRF. The Banerjee and Peikert PRF [BP14] is a more efficient lattice-based pseudorandom function. Let $0 < p < q$ be integer moduli. The public parameters consist of two uniformly and independently sampled matrices $\mathbf{A}_0, \mathbf{A}_1 \leftarrow \mathcal{R}_q^{m \times m\ell}$. Let $\mathbf{T}$ be a full binary tree, which is either a leaf or an ordered pair $(\mathbf{L}, \mathbf{R})$ of full binary trees. The message-mapping function $\mathbf{H}^{\text{bp}}$, parameterized by $\mathbf{T}$ and $\mathbf{A}_0, \mathbf{A}_1$, is defined as follows: For an input $x = (x_0, \dots, x_{n-1}) \in \{0, 1\}^n$,

$$\mathbf{H}^{\text{bp}}[\mathbf{T}](x) = \begin{cases} \mathbf{A}_{x_0} & \text{if } \mathbf{T} = \text{leaf and } |x| = 1; \\ \mathbf{H}^{\text{bp}}[\mathbf{L}](x_L) \cdot \mathbf{G}^{-1}(\mathbf{H}^{\text{bp}}[\mathbf{R}](x_R)) & \text{if } \mathbf{T} = (\mathbf{L}, \mathbf{R}) \text{ and } x = (x_L, x_R). \end{cases}$$

Let χ_s be a distribution over $\mathcal{R}$. We sample the secret key $\mathbf{k} \leftarrow \chi_s^m$. The BP14 PRF (parameterized by a full binary tree $\mathbf{T}$) is defined as

$$F_{\mathbf{k}}^{\text{bp}}(x) = \lfloor \mathbf{k}^\top \cdot \mathbf{H}^{\text{bp}}[\mathbf{T}](x) \rfloor_p \in \mathcal{R}_p^{m\ell}.$$

For suitable parameters, the resulting function family is pseudorandom under the decisional MLWE assumption [BP14]. The tree $\mathbf{T}$ determines the error growth and security level via its expansion $e(\mathbf{T})$ and sequentiality $s(\mathbf{T})$: For any leaf node leaf , $e(\text{leaf}) = s(\text{leaf}) \triangleq 0$; for an internal node $\mathbf{T} = (\mathbf{L}, \mathbf{R})$, we define

$$e(\mathbf{T}) \triangleq \max\{e(\mathbf{L}) + 1, e(\mathbf{R})\}, s(\mathbf{T}) \triangleq \max\{s(\mathbf{L}), s(\mathbf{R}) + 1\}.$$

That is, $e(\mathbf{T})$ and $s(\mathbf{T})$ are the maximum left and right depths of $\mathbf{T}$, respectively.

As mentioned in [BP14], the BLMR PRF can be viewed as the left spine version of BP14 PRF, where the public matrices correspond to the gadget-decompositions of dense matrices $\mathbf{A}_0, \mathbf{A}_1$ in BP14 PRF.

Zero-Knowledge Arguments of Knowledge. Let $R = \{(x, w)\}$ be a polynomial time decidable relation, where we call x the statement and w the witness. A *non-interactive zero-knowledge argument of knowledge* (NIZKoK) for R consists of three probabilistic polynomial-time algorithms:

- **Setup**(1^λ) = **crs**: Generates a common reference string.
- **Prove**(**crs**, x, w) = π : Generates a proof given input **crs**, and a pair $(x, w) \in R$.
- **Verify**(**crs**, x, π) = b : Verifies a proof π against a statement x , and outputs $b \in \{0, 1\}$.

A NIZKoK satisfies the following properties:

- *Completeness*: For every $(x, w) \in R$, an honestly generated proof is accepted except with negligible probability.
- *Knowledge soundness*: For every PPT prover $\mathcal{A}$ that produces an accepting proof with probability δ , there exists a probabilistic polynomial-time extractor $\mathcal{E}$ with black-box access to $\mathcal{A}$ such that, $\mathcal{E}$ extracts a witness w satisfying $(x, w) \in R$ with probability $\delta - \text{negl}(\lambda)$.
- *Zero knowledge*: There exists a probabilistic polynomial-time simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ that, without knowledge of the witness w , can simulate crs and proofs π whose distributions is computationally indistinguishable from that of honestly generated crs and proofs, respectively.

We will simplify NIZKoK as NIZK. We use the *POIP*-based approach to generate NIZK proofs, where an honest prover with input $(x, w) \in R$ proceeds in the following steps:

1. The prover and the verifier represent the relation R using low-degree polynomials.
2. Using a polynomial commitment scheme (PCS) to commit the multilinear extension of w , denoted by $\tilde{w}$;
3. Using arithmetic constraints to represent the relation R , and run a sumcheck protocol to prove it;
4. Using the PCS to evaluate the polynomial $\tilde{w}$.

Here we recall the definitions of multilinear extension and polynomial commitment scheme.

Multilinear Extension. Let $\mathbb{F}$ be a finite field and let $f : \{0, 1\}^\ell \rightarrow \mathbb{F}$ be a function defined over the Boolean hypercube. The *multilinear extension* (MLE) of f is the unique polynomial

$$\tilde{f} : \mathbb{F}^\ell \rightarrow \mathbb{F}$$

that has individual degree at most one in every variable and agrees with f over $\{0, 1\}^\ell$.

Polynomial Commitments. A PCS allows a prover to commit to a polynomial f and later prove evaluation of f at $\mathbf{z}$ without revealing f itself. It consists of algorithms (*Setup*, *Commit*, *Open*, *Verify*) with the following syntax:

$$\begin{aligned} \text{pp} &\leftarrow \text{Setup}(1^\lambda, \mathcal{P}), \\ (C, \text{aux}) &\leftarrow \text{Commit}(\text{pp}, f), \\ \pi &\leftarrow \text{Open}(\text{pp}, f, \text{aux}, \mathbf{z}, v), \\ b &\leftarrow \text{Verify}(\text{pp}, C, \mathbf{z}, v, \pi), \end{aligned}$$

where C is a commitment to f and $v = f(\mathbf{z})$. The proof π convinces the verifier that the polynomial committed in C evaluates to v at $\mathbf{z}$. A PCS satisfies:

- *Correctness*: an honestly generated opening for $v = f(\mathbf{z})$ is accepted except with negligible probability.
- *Binding*: after committing to a polynomial, a computationally bounded prover cannot open the same commitment inconsistently, except with negligible probability.

We can also define soundness and zero-knowledge of a PCS, which is very similar to the definition in NIZKoK.

3 Instantiating the VOPRF

In this section, we present a concrete instantiation of the LeOPaRd VOPRF protocol [ESTX26] using a variant of the BLMR PRF [BLMR13] and the corresponding NIZK proof systems. Formal definitions of VOPRF syntax, correctness, pseudorandomness, and request privacy are provided in Appendix A. For completeness, we first recall the construction of basic, round-optimal LeOPaRd protocol in online-only mode without preprocessing or batching. Next, we introduce our generalization of the BLMR PRF, where we group input bits into w -bit blocks to substantially reduce the evaluation depth and arithmetic complexity. Finally, we specify the formal NP relations for the NIZK proofs required to guarantee malicious security, and discuss why exact range proofs are necessary to avoid parameter blowup.

Server.Setup(1^λ) <i>// Initialize system parameters and master key pair.</i> 1. $\mathbf{k} \leftarrow \chi_k^m$; 2. $\text{ck}_i \leftarrow \text{COM.Setup}(1^\lambda)$ for $i \in \{1, 2, 3\}$; 3. $c_k \leftarrow \text{COM.Commit}(\text{ck}_2, \mathbf{k}; \rho_k)$; 4. Return $\text{pk} := (\text{ck}_1, \text{ck}_2, \text{ck}_3, c_k)$, $\text{sk} := (\mathbf{k}, \rho_k)$. Client.Request(pk, x) <i>// Mask client input x and generate NIZK proof π_c.</i> 1. $\mathbf{R} \leftarrow \chi_r^{h \times (\ell+m)}$; 2. $\mathbf{c}_r \leftarrow \text{COM.Commit}(\text{ck}_3, \mathbf{R}; \rho_r)$; 3. $\mathbf{V}_r := \text{RO}_r(\mathbf{c}_r) \in \mathcal{R}_q^{(\ell+m) \times m}$; 4. $\mathbf{B}_x := \bar{\mathbf{H}}(x)^\top \in \mathcal{R}_q^{h \times m}$; 5. $\mathbf{C}_x := \mathbf{R}\mathbf{V}_r + \mathbf{B}_x \pmod{q}$; 6. $\mathbf{d}_x \leftarrow \text{COM.Commit}(\text{ck}_1, x; \rho_x)$; 7. $\pi_c \leftarrow \text{NIZK}_1.\text{Prove}(\text{crs}_1, \text{stmt}_1, \text{wit}_1)$ where • $\text{stmt}_1 = (\mathbf{c}_r, \mathbf{C}_x, \mathbf{d}_x, \text{ck}_1, \text{ck}_3, \mathbf{V}_r, \bar{\mathbf{H}})$; • $\text{wit}_1 = (\mathbf{R}, x, \rho_r, \rho_x)$; 8. Return $\text{req} := (\mathbf{c}_r, \mathbf{d}_x, \mathbf{C}_x, \pi_c)$ to the server.	Server.BlindEval($\text{sk}, \text{pk}, \text{req}$) <i>// Verify π_c and blindly evaluate PRF under committed key $\mathbf{k}$.</i> 1. $\mathbf{V}_r := \text{RO}_r(\mathbf{c}_r)$; 2. If $\text{NIZK}_1.\text{Verify}(\text{crs}_1, \text{stmt}_1, \pi_c) = 0$, abort and return $\perp$; 3. $\mathbf{v} := \mathbf{V}_r \mathbf{k} + \mathbf{e}_s \pmod{q}$ for $\mathbf{e}_s \leftarrow \chi_e^{\ell+m}$; 4. $\mathbf{u} := \mathbf{C}_x \mathbf{k} + \mathbf{e}'_s \pmod{q}$ for $\mathbf{e}'_s \leftarrow \chi_1^h$; 5. $\pi_s \leftarrow \text{NIZK}_2.\text{Prove}(\text{crs}_2, \text{stmt}_2, \text{wit}_2)$ where • $\text{stmt}_2 = (\mathbf{u}, \mathbf{C}_x, \text{pk}, \text{ck}_2, \mathbf{v}, \mathbf{V}_r)$; • $\text{wit}_2 = (\mathbf{k}, \mathbf{e}_s, \mathbf{e}'_s, \rho_k)$; 6. Return $\text{resp} := (\mathbf{v}, \mathbf{u}, \pi_s)$ to the client. Client.Finalize($\text{pk}, \mathbf{R}, \text{req}, \text{resp}$) <i>// Verify π_s, unmask blinded response, and round.</i> 1. If $\text{NIZK}_2.\text{Verify}(\text{crs}_2, \text{stmt}_2, \pi_s) = 0$, abort and return $\perp$; 2. $\mathbf{w} := \mathbf{u} - \mathbf{R}\mathbf{v} + \text{RO}_{\text{pk}}(x, \text{pk}) \pmod{q}$; 3. $\mathbf{z} := \lfloor \mathbf{w} \rfloor_p \in \mathcal{R}_p^h$; 4. $y := \text{RO}_z(x, \mathbf{z}, \text{pk}) \in \{0, 1\}^\lambda$; 5. Return y.
---	---

Figure 1: The LeOPaRd VOPRF Protocol in online-only mode. We assume COM is a computationally binding and hiding commitment scheme, $\bar{\mathbf{H}}$ extracts the first h rows/columns of the message mapping $\mathbf{H}$, NIZK_1 and NIZK_2 are NIZK arguments for the relations in Figure 2, and $\text{RO}_r, \text{RO}_{\text{pk}}, \text{RO}_z$ are independent random oracles.

3.1 Overview of the VOPRF Protocol

The LeOPaRd protocol builds upon the ADDS blueprint [ADDs21], which to certain extent acts as a lattice-based analog of the classical 2HashDH construction [JKK14]. It is defined over the cyclotomic ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ with module ranks m (for the server) and ℓ (for the client). The server holds a master secret key $\mathbf{k} \in \mathcal{R}_q^m$ and publishes a public key commitment pk to $\mathbf{k}$. The client has a private input $x \in \{0, 1\}^n$ and wishes to learn the PRF output $y = \lfloor (p/q) \cdot (\mathbf{H}(x)^\top \mathbf{k}) \rfloor_p$ without revealing x to the server, while verifying that the server evaluated the function honestly using the key committed in pk .

To achieve obliviousness, the client blinds the message mapping matrix $\mathbf{B}_x = \bar{\mathbf{H}}(x)^\top \in \mathcal{R}_q^{h \times m}$ using a fresh MLWE sample, where $\bar{\mathbf{H}}$ is a truncated version of $\mathbf{H}$ for some dimension $h > 0$. The client first commits to its small masking randomness $\mathbf{R} \in \mathcal{R}_q^{h \times (\ell+m)}$ in $\mathbf{c}_r$, derives a pseudorandom matrix $\mathbf{V}_r = \text{RO}_r(\mathbf{c}_r) \in \mathcal{R}_q^{(\ell+m) \times m}$ via a random oracle, and computes the masked query $\mathbf{C}_x = \mathbf{R}\mathbf{V}_r + \mathbf{B}_x \pmod{q}$. By committing to $\mathbf{R}$ before querying the random oracle, the client cannot manipulate the distribution of $\mathbf{V}_r$ to bias the noise. The server blindly evaluates the request by multiplying $\mathbf{C}_x$ by $\mathbf{k}$ and adding small Gaussian noise $\mathbf{e}'_s$, alongside a public key share $\mathbf{v} = \mathbf{V}_r \mathbf{k} + \mathbf{e}_s \pmod{q}$. Thanks to the linear homomorphic structure of the masking, the client unmask the server response by computing $\mathbf{w} = \mathbf{u} - \mathbf{R}\mathbf{v} = \mathbf{B}_x \mathbf{k} + (\mathbf{e}'_s - \mathbf{R}\mathbf{e}_s) \pmod{q}$, and recovers the exact PRF output by rounding.

The complete protocol is formally described in Figure 1.

Client Relation $R_1[\mathbf{H}, \text{COM}]$: The client proves that its query $\mathbf{C}_x$ is computed honestly using private input x with the message mapping $\bar{\mathbf{H}}$ and short randomness $\mathbf{R}$:

$$R_1[\mathbf{H}, \text{COM}] = \left\{ \begin{array}{l} \text{stmt}_1 = (\mathbf{c}_r, \mathbf{C}_x, \mathbf{d}_x, \text{ck}_1, \text{ck}_3, \mathbf{V}_r, \bar{\mathbf{H}}) \\ \text{wit}_1 = (\mathbf{R}, x, \rho_r, \rho_x) \end{array} \middle| \begin{array}{l} \|\mathbf{R}\|_\infty \leq \beta_r \wedge \mathbf{B}_x = \bar{\mathbf{H}}(x)^\top \wedge \\ \mathbf{c}_r = \text{COM.Commit}(\text{ck}_3, \mathbf{R}; \rho_r) \wedge \\ \mathbf{d}_x = \text{COM.Commit}(\text{ck}_1, x; \rho_x) \wedge \\ \mathbf{C}_x = \mathbf{R}\mathbf{V}_r + \mathbf{B}_x \pmod{q} \end{array} \right\} \quad (4)$$

Server Relation $R_2[\text{COM}]$: The server proves that its response $\mathbf{u}, \mathbf{v}$ are computed honestly using the secret key $\mathbf{k}$ committed in pk :

$$R_2[\text{COM}] = \left\{ \begin{array}{l} \text{stmt}_2 = (\mathbf{u}, \mathbf{C}_x, c_k, \text{ck}_2, \mathbf{v}, \mathbf{V}_r) \\ \text{wit}_2 = (\mathbf{k}, \mathbf{e}_s, \mathbf{e}'_s, \rho_k) \end{array} \middle| \begin{array}{l} \|\mathbf{k}\|_\infty \leq \beta_k \wedge \|\mathbf{e}_s\|_\infty \leq \beta \wedge \|\mathbf{e}'_s\|_\infty \leq \beta_1 \wedge \\ c_k = \text{COM.Commit}(\text{ck}_2, \mathbf{k}; \rho_k) \wedge \\ \mathbf{v} = \mathbf{V}_r \mathbf{k} + \mathbf{e}_s \pmod{q} \wedge \mathbf{u} = \mathbf{C}_x \mathbf{k} + \mathbf{e}'_s \pmod{q} \end{array} \right\} \quad (5)$$

Figure 2: High-level NP Relations for Client and Server NIZK Arguments in LeOPaRd.

Correctness. During unmasking, the client computes

$$\begin{aligned} \mathbf{w} &= \mathbf{u} - \mathbf{R}\mathbf{v} + \text{RO}_{\text{pk}}(x, \text{pk}) \\ &= \mathbf{B}_x \mathbf{k} + \mathbf{e}_f + \text{RO}_{\text{pk}}(x, \text{pk}) \pmod{q}, \end{aligned} \quad (1)$$

where the accumulated evaluation noise is $\mathbf{e}_f = \mathbf{e}'_s - \mathbf{R}\mathbf{e}_s$. To ensure that $\lfloor \mathbf{w} \rfloor_p$ correctly recovers $\lfloor \mathbf{B}_x \mathbf{k} + \text{RO}_{\text{pk}}(x, \text{pk}) \rfloor_p$ with probability at least $1 - 2^{-\kappa}$, we must bound $\|\mathbf{e}_f\|_\infty \leq B_f$ relative to q/p . Assuming that the message mapping $\mathbf{H}$ satisfies ϵ_u -uniformity (Definition B.1 in Appendix B), correctness holds if

$$\frac{q}{p} \geq 2^{\kappa+2} \cdot hN \cdot (2B_f(\kappa, N) + 1). \quad (2)$$

In our implementation, the noise distributions χ_e and χ_1 are both discrete Gaussians with standard deviations s and s_1 , respectively, and the client randomness satisfies $\|\mathbf{R}\|_\infty \leq \beta_r$; so we bound B_f by:

$$B_f \leq \sqrt{\frac{\ln 2 \cdot (\kappa + 2 + \log_2(hN))}{\pi}} \cdot \left(s_1 + \beta_r s \sqrt{(\ell + m)N} \right). \quad (3)$$

We set $N = 512$, $m = \ell = 5$, $\beta_r = 1$, and $s_1 \approx 2^{17}$, ensuring $\kappa \geq 32$.

Security. We briefly restate the security properties of the LeOPaRd protocol. The reader can consult [ESTX26] for complete parameter constraints.

Theorem 3.1 (Pseudorandomness [ESTX26]). *Assume NIZK_1 is computationally sound, NIZK_2 is computationally zero-knowledge, COM is computationally hiding and extractable, the decisional MLWE problem is hard, and $\text{RO}_r, \text{RO}_{\text{pk}}, \text{RO}_z$ are independent random oracles. Then the LeOPaRd protocol in Figure 1 satisfies pseudorandomness against malicious clients.*

Theorem 3.2 (Request Privacy [ESTX26]). *Assume NIZK_1 is computationally zero-knowledge, NIZK_2 is computationally sound, the decisional MLWE problem is hard, $\text{RO}_r, \text{RO}_{\text{pk}}, \text{RO}_z$ are independent random oracles, and the error probability $p_u = hN(2B_f + 1)p/q$ is negligible. Then the LeOPaRd protocol in Figure 1 satisfies request privacy against malicious servers.*

3.2 High-Level NIZK Relations

To achieve malicious security, both parties must generate NIZK proofs certifying the well-formedness of their messages. The high-level relations are formally specified in Figure 2.

3.3 BLMR PRF Optimization

We instantiate the LeOPaRd protocol using the BLMR PRF with some tweaks to reduce online latency and make the proof systems more efficient. In particular, our instantiation uses 2^w public random matrices to define the message mapping $H^{\text{blmr}}(\cdot)$ for $w \geq 1$, where $w = 1$ corresponds to the original BLMR definition. Tuning the parameter w provides an interesting optimization opportunity for the online latency and the size of the required NIZK proof, at the cost of potentially larger memory footprint. We will discuss concrete choices of w in later sections. Below we present such modified PRF and its security property.

Definition 3.3 (BLMR+ message mapping). Let $w, k \geq 1$ be integers, and let $\{\mathbf{A}_{\mathbf{h}} \in \mathcal{R}_q^{m\ell \times m\ell} \mid \mathbf{h} \in \{0, 1\}^w\}$ be a list of binary matrices labeled by bitstrings of length w . Given a bitstring $\mathbf{x} \in \{0, 1\}^{wk}$, parse $\mathbf{x}$ as a concatenation $\mathbf{x} = \mathbf{x}_0\mathbf{x}_1 \dots \mathbf{x}_{k-1}$ where $\mathbf{x}_i \in \{0, 1\}^w$ for $i \in [k]$. Then the BLMR+ message mapping $H^{\text{blmr}} : \{0, 1\}^{wk} \rightarrow \mathcal{R}_q^{m\ell \times m\ell}$ is defined as

$$H^{\text{blmr}}(\mathbf{x}) = \prod_{i=0}^{k-1} \mathbf{A}_{\mathbf{x}_i}. \quad (6)$$

In the OPRF protocol, we use a truncated message mapping $\bar{H}^{\text{blmr}} : \{0, 1\}^{kw} \rightarrow \mathcal{R}_q^{h \times m\ell}$ where $\bar{H}^{\text{blmr}}(\mathbf{x}) = H^{\text{blmr}}(\mathbf{x})[1 : h]$ consisting of the first h rows of $H^{\text{blmr}}(\mathbf{x})$.

Definition 3.4 (BLMR+ PRF). Let $w, k \geq 1$ be integers, and let $\mathbb{A} = \{\mathbf{A}_{\mathbf{h}} \in \mathcal{R}_q^{m\ell \times m\ell} \mid \mathbf{h} \in \{0, 1\}^w\}$ be matrices over $\mathcal{R}_q$. Define the function family, parameterized by matrices in $\mathbb{A}$ and a modulus $p < q$:

$$\mathcal{F}_{\mathbb{A}, p} \triangleq \{F_{\mathbf{k}} : \{0, 1\}^{kw} \rightarrow \mathcal{R}_q^{m\ell} \mid \mathbf{k} \in \mathcal{R}_q^m\},$$

where the functions $F_{\mathbf{k}}$ are defined as

$$F_{\mathbf{k}}(\mathbf{x}) \triangleq \lfloor \mathbf{k}^\top \cdot H^{\text{blmr}}(\mathbf{x}) \rfloor_p.$$

As in [BLMR13], the above BLMR+ construction is a secure PRF family assuming the $\text{MLWE}_{m, \mathcal{R}_q, \mathcal{R}_q, \chi_e}$ problem is hard and $q/p \geq 2^k \cdot (2B_{\chi_e}m^k + 1)m \cdot 2^\lambda$ for some high probability bound B_{χ_e} on the ℓ_∞ norm of samples from χ_e . Note that such bound on the ratio q/p is only to account for the accumulated noise that is injected during the security proof. In our implementation we will use heuristic parameters which satisfy the concrete security of the underlying MLWE problem but would cause the accumulated noise size to be close to the ratio q/p ; we believe such parameters still provide the desired PRF security, while the large accumulated noise is only an artifact of the proof technique.

Public Matrix Generation. To save bandwidth, the public matrices $\{\mathbf{A}_{\mathbf{h}}\}_{\mathbf{h} \in \{0, 1\}^w}$ are generated deterministically from independent 256-bit random seeds using a PRG. During evaluation, the matrices $\mathbf{A}_{\mathbf{x}_i}$ are expanded on-the-fly and cached in memory ($2^8 = 256$ matrices require roughly 8 MB of memory).

4 Constructions of Zero-Knowledge Proofs

In this section, we discuss how to construct the NIZK proof systems for the relations in Figure 2. We focus on the client's proof π_c for Relation R_1 as it encompasses the non-linear PRF evaluation, exact norm proofs for the commitment randomness, and commitment verifications. For better readability, we recall the NP relation R_1 from Section 4.1 and then explain how to construct the corresponding arithmetic constraints over the cyclotomic ring $\mathcal{R}_q$ and a finite field. In Section 4.2, we describe the ring-switching techniques for converting constraints over $\mathcal{R}_q$ into constraints on low-degree polynomials over the extension field $\mathbb{F}_{q^d}$. We then prove the resulting constraints over the finite field using the sumcheck protocol together with a polynomial commitment scheme. In Section 4.3, we discuss how to make these sumcheck protocols zero-knowledge. Finally, in Section 4.4, we discuss the server's proof π_s for Relation R_2 , which is comparatively lightweight and can be implemented with a scaled down version of the client's proof.

4.1 Arithmetizing the NP Statements

Recall Relation R_1 from Section 3.2 (see also [ESTX26, Figure 10]). Let COM be the lattice commitment scheme instantiated with commitment keys ck_1, ck_3 and norm bound β_r . Let $\mathbf{V}_r \triangleq \text{RO}(\mathbf{c}_r)$ be the matrix derived from a random oracle over commitment $\mathbf{c}_r$. Given the statement $(\mathbf{V}_r, \mathbf{C}_x, \mathbf{c}_r)$, the prover aims to prove knowledge of a short matrix $\mathbf{R}$ and a bit string $x \in \{0, 1\}^n$ satisfying

$$\mathbf{C}_x = \mathbf{R}\mathbf{V}_r + \mathbf{B}_x \pmod{q} \quad \text{and} \quad \|\mathbf{R}\|_\infty \leq \beta_r,$$

where $\mathbf{B}_x \triangleq \bar{\mathbf{H}}^{\text{blmr}}(x)$. For the full definition of R_1 , see Figure 2.

This NP relation is very similar to that in [ESTX26]. The main difference is that we instantiate the message mapping function with a variant $\mathbf{H}^{\text{blmr}}$ of the construction from Boneh-Lewi-Montgomery-Raghuathan [BLMR13], as explained in Section C.

Recall that $|x| = n$ and $k = n/w$. In our construction, the input x is partitioned into k blocks $\mathbf{x}_0, \dots, \mathbf{x}_{k-1}$, each consisting of w bits, and the public parameter consists of $k \cdot 2^w$ matrices $\mathbf{A}_{i,j} \in \mathcal{R}_q^{m_\ell \times m_\ell}$, where each polynomial entry of $\mathbf{A}_{i,j}$ is invertible, and has coefficients sampled independently and uniformly from $\{0, 1\}$, for $i \in [k], j \in [2^w]$. Our goal is to represent the computation of $\bar{\mathbf{H}}^{\text{blmr}}$ using arithmetic constraints over $\mathcal{R}_q$. To this end, we define variables $\mathbf{B}_i \in \mathcal{R}_q^{m_\ell}$ for $i \leq k-2$, such that

- $\mathbf{B}_0 \triangleq \mathbf{A}_{0,\mathbf{x}_0}[0]$,
- $\mathbf{B}_i \triangleq \mathbf{A}_{i,\mathbf{x}_i} \cdot \mathbf{B}_{i-1}$ for $i = 1, \dots, k-2$, and
- $\bar{\mathbf{H}}^{\text{blmr}}(x) = \mathbf{B}_x = \mathbf{A}_{k-1,\mathbf{x}_{k-1}} \cdot \mathbf{B}_{k-2}$.

Again, all above computation carry out over the ring $\mathcal{R}_q$.

Roughly speaking, $\mathbf{B}_i$ captures an intermediate computation in the evaluation of $\bar{\mathbf{H}}^{\text{blmr}}$. To arithmetize $\mathbf{B}_i = \mathbf{A}_{i,\mathbf{x}_i} \cdot \mathbf{B}_{i-1}$, we introduce a witness $e_i \in \{0, 1\}^{2^w}$ that serves as the indicator vector of $\mathbf{x}_i$. Specifically, for each $j \in [2^w]$, $e_{i,j} = 1$ if $\mathbf{x}_i$ is the binary representation of j , and $e_{i,j} = 0$ otherwise. Then, proving $\mathbf{B}_i = \mathbf{A}_{i,\mathbf{x}_i} \cdot \mathbf{B}_{i-1}$ can be reduced to proving

$$\begin{aligned} \mathbf{B}_i &= \sum_{j \in [2^w]} e_{i,j} \cdot \mathbf{A}_{i,j} \cdot \mathbf{B}_{i-1}, \\ \forall j \in [2^w] : e_{i,j} &\in \{0, 1\}, \\ 1 &= \sum_{j \in [2^w]} e_{i,j}. \end{aligned} \tag{7}$$

Here, we note that the last two constraints together ensure that e_i is a valid indicator vector.

Bit check. If we simply follow the proof system in LeOPaRd, one main challenge in proving Equation (7) is to show that $e_{i,j} \in \{0, 1\}$. This is because [ESTX26] adopts a LaBRADOR-style proof system. LaBRADOR proves constraints directly over $\mathcal{R}_q$, and it is hard to construct a constraint over $\mathcal{R}_q$ to capture that $e_{i,j} \in \{0, 1\}$.

On the other hand, if we design and prove the constraints over a finite field, following the original work of [ADDS21], bit checks can be captured by the low-degree polynomial constraint $e_{i,j} \cdot (1 - e_{i,j}) = 0$. However, proving the ring constraints, i.e., $\mathbf{B}_i = \sum_{j \in [2^w]} e_{i,j} \cdot \mathbf{A}_{i,j} \cdot \mathbf{B}_{i-1}$, then becomes expensive.

This dilemma has been a challenge in previous proof systems. Looking ahead, we will address it using the ring-switching technique, which efficiently supports and bridges these two types of constraints. After the ring-switching step, all constraints are over finite fields, and the bit checks can be simply enforced by the constraint $e_{i,j} \cdot (1 - e_{i,j}) = 0$.

Unlike LeOPaRd [ESTX26] which uses a LaBRADOR-style approximate range proof, our proof system proves exact norm bounds.

Proof of the commitment. In addition to proving the hash evaluation, recall that in Relation R_1 , we also need to prove the wellformedness of commitments. Namely, for public $(\mathbf{c}_r, \mathbf{C}_x, \mathbf{d}_x)$, the prover must prove knowledge of binary x and short $\mathbf{R}, \rho_r, \rho_x$ such that

- $\mathbf{c}_r = \text{COM.Commit}(\text{ck}_3, \mathbf{R}; \rho_r)$;
- $\mathbf{d}_x = \text{COM.Commit}(\text{ck}_1, x; \rho_x)$; and
- $\mathbf{C}_x = \mathbf{R} \cdot \mathbf{V}_r + \mathbf{B}_x$.

Here, ck_1 and ck_3 are the commitment keys, containing uniformly random $\mathbf{A}$ and $\mathbf{b}$.

Following [ESTX26], we instantiate the commitment using a Regev-style commitment. See Section B for detailed definition of COM. Such commitment scheme can be arithmetized using constraints over $\mathcal{R}_q$. We use an exact ℓ_∞ -norm proof to prove shortness, and the arithmetization is very similar to that of the bit check. These constraints can be expressed as

$$\begin{aligned} \mathbf{c}_r &= (\mathbf{A}_3 \rho_r, b_m \langle \mathbf{b}_{3,0}, \rho_r \rangle + \mathbf{R}_0, b_m \langle \mathbf{b}_{3,1}, \rho_r \rangle + \mathbf{R}_1, \dots), \\ \mathbf{d}_x &= (\mathbf{A}_1 \rho_x, b_m \langle \mathbf{b}_{1,0}, \rho_x \rangle + e'_0, b_m \langle \mathbf{b}_{1,1}, \rho_x \rangle + e'_1, \dots). \end{aligned} \tag{8}$$

$$\begin{aligned} \|\mathbf{R}\|_\infty &\leq \beta_r, \\ \|\rho_r\|_\infty &\leq \beta_r, \\ \|\rho_x\|_\infty &\leq \beta_r, \end{aligned}$$

where $b_m = 2$ is the message space of COM. As in [ESTX26], we set $\beta_r = 1$, allowing us to impose the polynomial constraint $(1 - X) \cdot (1 + X) \cdot X = 0$.

Putting things together. By combining above constraints, our witness becomes $(\mathbf{B}_0, \dots, \mathbf{B}_{k-2}, e_0, \dots, e_{k-1}, \mathbf{R}, \rho_r, \rho_x)$, and we prove that it satisfies three groups of constraints: the PRF evaluation constraints in Equation (7), and the commitment relations in Equation (8).

Lastly, we emphasize that these arithmetic constraints are defined over two different algebraic structures: $\mathcal{R}_q$ and a finite field. We discuss how to bridge this gap in the next subsection.

4.2 Proving the Constraints via Sumcheck

The arithmetization process defined in Section 4.1 mainly introduces two types of constraints:

- Ring constraints, such as $\mathbf{B}_i = \sum_j e_{i,j} \cdot (\mathbf{A}_{i,j} \cdot \mathbf{B}_{i-1})$, and
- Range checks, such as $(1 - e_{i,j}) \cdot e_{i,j} = 0$.

The first type of constraint is defined over $\mathcal{R}_q$, while the second type is defined over finite fields. Therefore, neither standard sumcheck protocols, which operate only over finite fields, nor LaBRADOR-style proofs, which mainly operate over $\mathcal{R}_q$, natively apply to our setting. To bridge this gap, we use the ring-switching technique of Huang, Mao, and Zhang [HMZ26], and we briefly review it below.

Ring switching [HMZ26]. The core idea of the ring-switching is to convert constraints over the ring $\mathcal{R}_q$ to constraints over a finite field. It consists of two steps.

First, observe that a ring relation such as

$$\mathbf{B}_i = \sum_j e_{i,j} \cdot (\mathbf{A}_{i,j} \cdot \mathbf{B}_{i-1})$$

over $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ holds if and only if there exists a vector of polynomials $\mathbf{r}_i$ of degree at most $N - 1$ such that

$$\mathbf{B}_i = \sum_j e_{i,j} \cdot (\mathbf{A}_{i,j} \cdot \mathbf{B}_{i-1}) + (X^N + 1) \cdot \mathbf{r}_i.$$

The latter relation is defined over $\mathbb{Z}_q[X]$, i.e., without taking the quotient by $(X^N + 1)$. Here the vector $\mathbf{r}_i$ contains polynomials of degree at most $N - 2$, and it is treated as an additional witness.

Assume q is a prime number for simplicity. Since q is usually not very large in lattice-based constructions, we further use an extension field and view these constraints as being defined over $\mathbb{F}_{q^d}[X]$ to ensure sufficient soundness for the sumcheck protocol. This step is called *ring embedding* in [HMZ26].

However, performing sumcheck directly over such a large ring is expensive. The second step of ring-switching further collapses the constraints to $\mathbb{F}_{q^d}$, a much smaller finite field. The idea is simple: Notice that each entry of $\mathbf{B}_i$, $\mathbf{A}_{i,j}$, and $\mathbf{r}_i$ is a polynomial over $\mathbb{F}_{q^d}[X]$ of degree at most N . The verifier randomly samples a challenge $\alpha \in \mathbb{F}_{q^d}$, and both the prover and the verifier then reduce these polynomials by evaluating them at $X = \alpha$. Formally,

1. The verifier sends a random challenge $\alpha \in \mathbb{F}_{q^d}$.
2. The prover and the verifier then prove the following statement:

$$\mathbf{B}_i(\alpha) = \sum_j e_{i,j}(\mathbf{A}_{i,j}(\alpha)\mathbf{B}_{i-1}(\alpha)) + (\alpha^N + 1)\mathbf{r}_i(\alpha).$$

Here $\mathbf{A}_{i,j}(\alpha)$ and $\mathbf{B}_i(\alpha)$ denote the matrices obtained by evaluating each of their entries, viewed as polynomials, at $X = \alpha$.

Since $e_{i,j}$ is constant polynomial, it does not need to be evaluated at α . After these two steps, the prover and verifier reduce the original constraints to constraints over $\mathbb{F}_{q^d}$.

On the other hand, the range-proof-related constraints are already defined over $\mathbb{F}_{q^d}$. Therefore, we can safely use constraints such as $X \cdot (1 - X) = 0$ to perform bit checks.

Due to space limitations, we leave ring switching as a high-level discussion. Readers can refer to [HMZ26] for more details.

Soundness of ring switching. Since the entries of $\mathbf{B}_i$, $\mathbf{A}_{i,j}$, and $\mathbf{r}_i$ are polynomials of degree at most N , the soundness error of the ring-switching step is at most $N/|\mathbb{F}_{q^d}| = N/q^d$. In our implementation, we choose q to be a 64-bit prime following the suggestion of [ESTX26], and hence we set $d = 2$ to ensure sufficient soundness.

Multilinear extensions. Both types of constraints are now relations over $\mathbb{F}_{q^d}$. We can therefore use a standard sumcheck-based approach to complete the NIZK proof.

Recall that the witness includes $(\mathbf{B}_i, e_i, \mathbf{r}_i, \mathbf{R}, \rho_r, \rho_x)_{i \in [k]}$. We use a multilinear polynomial to commit to the coefficients of each entry of the witness, and then compile the constraints into multilinear extensions.

For example, for each $i \in [k]$, the witness $\mathbf{r}_i$ is a polynomial $\mathbf{r}_i = \sum_{l \in [N]} r_{i,l} \cdot X^l$. We define its multilinear extension $\tilde{\mathbf{r}}$ such that, for each $i \in [k]$ and $l \in [N]$:

$$\tilde{\mathbf{r}}(i, l) = r_{i,l}.$$

We can similarly define $\tilde{\mathbf{B}}$ and $\tilde{e}$.

For the public parameters, we take the multilinear extensions of their evaluations at α . For example, we define $\tilde{\mathbf{A}}$ as the multilinear extension of $\mathbf{A}(\alpha)$. Finally, by defining $\tilde{\alpha}(l) \triangleq \alpha^l$ for $l \in [N]$, we can express our constraints as low-degree polynomial identities, such as:

$$\begin{aligned} \sum_{l \in [N]} \tilde{\mathbf{B}}(i, l) \tilde{\alpha}(l) &= \sum_{j \in [2^w]} \tilde{e}(i, j) \tilde{\mathbf{A}}(i, j) \sum_{l \in [N]} \tilde{\mathbf{B}}(i-1, l) \tilde{\alpha}(l) \\ &\quad + (\alpha^N + 1) \sum_{l \in [N]} \tilde{\mathbf{r}}(i, l) \tilde{\alpha}(l). \end{aligned}$$

We can now represent all the constraints using low-degree polynomials. This is the standard form required before applying the sumcheck protocol.

Proving the constraints via sumcheck. The final step of our NIZK proof is to batch all of these polynomial constraints into a single polynomial. Following standard approaches such as Spartan [Set20], we use the equality function $\tilde{eq}$ to batch the constraints, so that all constraints hold if and only if

$$\begin{aligned} H_\alpha(\tau_\alpha) \triangleq \sum_{i \in [k]} \tilde{eq}(\tau_\alpha, i) & \left(\sum_{j \in [2^w]} \tilde{e}(i, j) \tilde{\mathbf{A}}(i, j) \sum_{l \in [N]} \tilde{\mathbf{B}}(i-1, l) \tilde{\alpha}(l) \right. \\ & + (\alpha^N + 1) \sum_{l \in [N]} \tilde{\mathbf{r}}(i, l) \tilde{\alpha}(l) \\ & \left. - \sum_{l \in [N]} \tilde{\mathbf{B}}(i, l) \tilde{\alpha}(l) \right) \end{aligned}$$

is zero. Similarly, for the norm checks, the prover proves that

$$H_0(\tau_0) \triangleq \sum_{i \in [k]} \sum_{l \in [N]} \tilde{eq}(\tau_0, (i, l)) \tilde{e}(i, l) (\tilde{e}(i, l) - 1)$$

is zero, and the relation also works for $\mathbf{R}$, ρ_r , ρ_x . The verifier samples random challenges τ_α and τ_0 , and the prover then proves $H_\alpha(\tau_\alpha) = H_0(\tau_0) = 0$ using the sumcheck protocol.

Each sumcheck terminates by reducing its claim to evaluations of the witness and $\tilde{\mathbf{r}}$ at random points, which the prover opens using the PCS employed to commit to them.

Note that the equations presented in this section do not constitute the full constraint system. Rather, they demonstrate how we transform relations over $\mathcal{R}_q$, which cannot be handled directly by sumcheck, into sums over a Boolean cube of low-degree polynomials in the committed multilinear extensions. The remaining constraints differ only in their public coefficients.

Although the statement being proved is a lattice-based one, the argument itself is not. Its challenges are drawn from $\mathbb{F}_{q^d}$ rather than from $\mathcal{R}_q$, and its soundness relies on the sumcheck protocol and the binding property of the commitment scheme.

4.3 Zero Knowledge

Sumcheck protocols are not natively zero-knowledge. However, techniques for converting them to be zero-knowledge have been extensively studied. See [Tha22] for a survey. The overall proof consists of two components: the sumcheck transcript itself and the transcript of the PCS, both of which need to be made zero-knowledge.

Masking the sumchecks. To make the sumcheck transcript zero-knowledge, we use a standard masking technique. Specifically, we use the framework of Libra [XZZ⁺19]. All the required masks are stored in the multilinear extension of the witness, using the indices introduced when padding the witness. Therefore, they require no additional commitments and do not increase the number of sumcheck variables.

Zero-knowledge PCS. We can instantiate using any zero-knowledge PCS. In our implementation, however, we use Akita [DBK⁺26], a recent lattice-based high-performance and versatile PCS but currently does not support zero-knowledge yet. As far as we know, the zero-knowledge support is being actively developed, and once it is available, we can integrate it into our implementation to obtain a fully zero-knowledge protocol. In our performance estimation in Section 5, we estimated the ZK overhead according to standard techniques.

4.4 Server Proof

Now, recall that as part of blind evaluation of a client request, the server needs to prove in π_s Relation R_2 :

$$\left(\begin{array}{l} ((\mathbf{u}, \mathbf{C}_x, c_k, \text{ck}_2, \mathbf{v}, \mathbf{V}_r) \\ (\mathbf{k}, \mathbf{e}_s, \mathbf{e}'_s, \rho_k)) \end{array} \left| \begin{array}{l} \|\mathbf{k}\|_\infty \leq \beta_k \wedge \|\mathbf{e}_s\|_\infty \leq \beta \\ \wedge \|\mathbf{e}'_s\|_\infty \leq \beta_1 \\ \wedge c_k = \text{COM.Commit}(\text{ck}_2, \mathbf{k}; \rho_k) \\ \wedge \mathbf{v} = \mathbf{V}_r \mathbf{k} + \mathbf{e}_s \\ \wedge \mathbf{u} = \mathbf{C}_x \mathbf{k} + \mathbf{e}'_s \end{array} \right. \right)$$

We note that the constraints in R_2 are all covered by those in R_1 . However, unlike R_1 , Relation R_2 does not need to prove $\bar{\mathbf{H}}^{\text{blmr}}$. Therefore, in the arithmetization step of R_2 , we do not need to introduce any intermediate variables (which also become part of the witness), such as $\mathbf{B}_i$. In our concrete parameter set, the total number of field elements in the witness of R_2 is upper-bounded by 2^{15} , which is far smaller than the number of field elements in the witness of R_1 , namely 2^{18} field elements (see Section 5.3). Therefore, the overall running time for proving R_2 is much lower than that for proving R_1 .

5 Experimental Results

To demonstrate the concrete performance of our protocol, we developed a prototype implementation of Beasley in Rust. The source code is available at:

<https://github.com/LatticeProofs/Beasley>

Our implementation currently covers the client-side evaluation of the message mapping $\bar{\mathbf{H}}^{\text{blmr}}$, the ring-switching transformation, and the resulting sumcheck protocol described in Section 4. All benchmarks were conducted on an Intel Core i5-13600KF CPU with AVX2 enabled, running on a single core. The measured part takes 46.73 ms to prove, 9.79 ms to verify, and produces a 2.9 KB transcript. We do not implement the PCS ourselves. Instead, we use a recent open-source Akita PCS [DBK⁺26] and benchmarked on the same hardware. Based on our benchmarks of Akita, we estimate that a complete client request generation takes 519.84 ms, server verification takes 15.29 ms, with a peak RAM footprint of 113.7 MB and a total proof size of 75.4 KB.

5.1 Implementation Details

We follow the parameters suggested by [ESTX26], except that we use a larger ring with smaller module ranks for MLWE. Specifically, we choose the cyclotomic ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ with $N = 512$ and module rank $m = 5$ for both the client and the server. In contrast, [ESTX26] suggests $N = 64$, $m_s = 37$ (for server), and $m_c = 40$ (for client), because its NIZK proof system, LaBRADOR, favors a smaller ring dimension. We choose $q = 2^{64} - 59$ because this modulus is supported by Akita [DBK⁺26]. The message mapping $\bar{\mathbf{H}}$ takes a 128-bit input, i.e., $|x| = n = 128$, also following [ESTX26], except that x is divided into $k = 32$ blocks of $w = 4$ bits each. As in LeOPaRd, our parameters achieve 128 bits computational security for 2^{32} queries. A complete list of parameters is provided in Table 2.

Integer operations and data structures. Field elements are stored as unsigned 64 bit integers for compatibility with `tfhe-ntt`. Sampling in $\mathbb{F}_q$ assembles 64 bits from a PRNG and rejects values that are at least q . Polynomials are represented as coefficient slices over `u64`. Vectors and matrices of polynomials are stored as tightly packed coefficients in a single vector (Array of Structure format). Decomposition of vectors of polynomials is done by sequentially decomposing coefficients. The two commitment keys are drawn from SHAKE128, both are generated once at setup, and NTT of the commitment keys are precomputed at the same time. Setup is timed separately and is excluded from Table 3.

Table 2: Concrete parameters for our implementation.

Parameter	Description	Value
q	Prime modulus	$2^{64} - 59$
N	Ring dimension of $\mathcal{R}_q$	512
m	Module rank	5
d	Degree of extension field $\mathbb{F}_{q^d}$	2
n	Input length of PRF, in bits	128
w	Number of bits per input block	4
k	Number of input blocks (n/w)	32
β_r	ℓ_∞ bound on client randomness $\mathbf{R}$	1

Table 3: Cost of the complete client proof for Relation R_1 . The measured row corresponds to Section 4.2 without the PCS, benchmarked on an Intel Core i5-13600KF with AVX2 using a single core, with the median reported over 50 runs. The estimated row is based on our Akita benchmarks obtained on the same hardware.

Component	Cost
<i>Client time (ms)</i>	
Request generation	24.34
Sumcheck prover	22.39
PCS prover (est.)	473.11
Total	519.84
<i>Server verification time (ms)</i>	
Sumcheck verifier	9.79
PCS verifier (est.)	5.5
Total	15.29
<i>Communication (KB)</i>	
Request and sumcheck proof	2.9
PCS proof (est.)	72.5
Total	75.4
<i>Peak RAM (MB)</i>	
Request generation and sumcheck	113.7
PCS prover (est.)	74

Field Arithmetic. We use the extension field $\mathbb{F}_{q^2}$ to ensure approximately 128 bits of security. A multiplication in the base field $\mathbb{Z}_q$ forms a 128-bit product, the modulo of q only costs shift and addition because $2^{64} \equiv 59 = 2^6 - 2^2 - 1$. Since $q \equiv 5 \pmod{8}$, 2 is a quadratic non-residue and $X^2 - 2$ is irreducible, so we may take $\mathbb{F}_{q^2} = \mathbb{F}_q[X]/(X^2 - 2)$. The folding constant is then 2, and reducing X^2 costs a shift rather than a multiplication. Arithmetic in $\mathbb{F}_{q^2}$ dominates the sumchecks and hence the prover, so the multiplication of field elements in $\mathbb{F}_{q^2}$ are optimized and vectorised with AVX2.

Ring Arithmetic. Beasley requires general polynomial arithmetic over $\mathcal{R}_q$, as well as over $\mathbb{Z}_q[X]$ (to transition to the sumcheck). Our arithmetic library builds on top of the **tfhe-ntt** library, which provides SIMD (AVX-2 or AVX-512) optimized NTT-multiplication. Since the prime modulus q is not NTT-friendly (due to the requirement of PCS), we cannot use NTT-based multiplication directly for $\mathcal{R}_q$. Instead, we use RNS to transform to 3 sets of 32-bit NTT provided by **tfhe-ntt** and reconstruct by CRT. In Beasley, multiplication over $\mathcal{R}_q$ is always required with one operand being bits, so a coefficient of the linear convolution is at most $(q - 1) \cdot N \cdot m$ and 3 primes suffice.

As in Akita [DBK⁺26] we avoid reducing modulo $X^N + 1$ inside the transform by padding. A product of two polynomials of degree $< N$ has degree $< 2N - 1$, we therefore pad the polynomials to degree $2N - 1$ and

Table 4: Estimated cost of the PCS. The first column shows Akita’s running time for a polynomial of size 2^{18} . The second column provides an estimate of the ZK overhead.

Operation	Akita (2^{18})	With ZK Overhead
Commit	28.4 ms	31.24 ms
Open	401.7 ms	441.87 ms
Verify	5 ms	5.5 ms
Proof Size	62.5 KB	72.5 KB

use the existing library to efficiently multiply over $\mathbb{Z}_q[X]/(X^{2^N} + 1)$. The result does not incur any reduction modulo $X^{2^N} + 1$ and so holds over $\mathbb{Z}_q[X]$. Given the full convolution, we perform optimized polynomial division to compute the residues modulo $(X^N + 1)$.

We minimize the number of NTTs that must be performed by keeping re-used and intermediate values in NTT form.

Optimized First Round Sumcheck. A sumcheck round folds the table by interpolating between pairs of coefficients at the round challenge which in general costs a field multiplication per pair. Since some of the coefficients of the witness are bits, in the first round the fold and the round polynomial reduce to selections and additions for the binary part of the witness. Since only the first round handles the whole table (each subsequent round works on half as many coefficients), it accounts for a large portion of the folding work.

Zero-knowledge masks. The masks of Section 4.3 are field elements, but they must be committed before the challenges are drawn. Rather than commit to them separately, we store each mask into the index that the witness had to pad out for multilinear extension anyway. Therefore, the number of sumcheck variables is unchanged. The cost of zero knowledge lands almost entirely in the transcript, and prover time shows no measurable increase.

5.2 Performance of Sumcheck Proofs

The concrete running time and proof size for the client NIZK of Beasley, excluding the PCS, are shown in Table 3. Two components dominate the prover time: folding the 2^{18} coefficients of the witness in the batched sumcheck takes 24% of the total time, while constructing the witness and the public tables accounts for another 58%, most of which is spent evaluating H itself. Evaluating the public tables at α accounts for 18%, and everything else combined takes less than 1%.

5.3 PCS Cost Estimation

To estimate the PCS cost in our proof systems, we run Akita [DBK⁺26] on the same hardware as our own benchmarks. Akita does not currently support zero knowledge. We therefore apply an adjustment for using standard techniques to achieve ZK when estimating the cost.

Rationale for ZK overhead estimation. Although Akita currently lacks zero-knowledge support, there is no evidence of any inherent barrier. Based on private discussion with the Akita team, ZK-Akita will be available in the near future.

Making proof systems zero-knowledge has been an active research and engineering topic recently. For example, a ZK version of the hash-based WHIR PCS [ACFY25] has been implemented¹ approximately one year after the original WHIR paper was published. The LaZer team recently made LaBRADOR zero-knowledge [BBL⁺26], approximately one year after releasing the original non-ZK version of LaBRADOR [LSS24].

¹<https://github.com/Plonky3/Plonky3/issues/1590>

Both implementations rely on well-established techniques, and the resulting ZK overhead is relatively modest. For reference, making LaBRADOR zero-knowledge increases the proof size by only 10 KB [BBL⁺26]. Akita could similarly adopt the standard techniques to achieve zero-knowledge.

Based on the ZK overhead observed in WHIR and LaBRADOR, we use a similar estimate for Akita. Specifically, we estimate a 10% increase in running time and an additional 10 KB in proof size.

Running time. We commit the witness $(\mathbf{B}_i, e_i, \mathbf{r}_i, \mathbf{R}, \rho_r, \rho_x)$ using one multilinear PCSs. The first part, $(e_i, \mathbf{R}, \rho_r, \rho_x)$, contains coefficients that are bits; the second part, $(\mathbf{B}_i, \mathbf{r}_i)$, contains coefficients that are full-range elements of $\mathbb{F}_q$.

For a polynomial with 2^{18} coefficients over a 64-bit modulus, Akita achieves commitment, proof generation, and verification times of 28.4 ms, 401.7 ms, and 5 ms, respectively.

We then apply an additional 10% running time to account for the estimated zero-knowledge overhead. We therefore estimate the PCS costs are 31.24 ms to commit, 441.87 ms to open, and 5.5 ms to verify. Together with our sumcheck benchmarks, a complete proof of R_1 takes about **520** ms to produce and at most **15** ms to verify.

Proof size. For the opening proof, we directly use the proof size reported by Akita [DBK⁺26], where a polynomial with 2^{18} coefficients over a 64-bit modulus is decomposed into $7 \cdot 2^{18}$ 10-bit digits, resulting in an opening proof of size 62.56 KB. Finally, we add a further 10 KB to account for the estimated zero-knowledge overhead and take **72.56** KB as a conservative bound on the opening proof size. Together with Table 3, a complete client proof has a size of about **75.4** KB.

Peak memory. In Akita, the measured peak RAM usage is 64 MB for a polynomial with 2^{18} coefficients, by charging the same 10% for zero knowledge gives 74 MB. Peak memory is not additive; rather, it is the maximum memory usage over time. Since the two steps can be run sequentially, we report the larger of the two peak memory usages, namely 113.7 MB.

Summary. A complete proof of Relation R_1 costs 519.84 ms to produce, at most 15.29 ms to verify, and has a size of 75.4 KB. Of these costs, 46.73 ms, 9.79 ms, and 2.9 KB respectively are measured directly, while the remaining costs are estimated from Akita [DBK⁺26] benchmarks obtained on the same hardware. The final costs of the complete client proof are shown in Table 3.

6 Conclusion and Future Works

Beasley achieves an order-of-magnitude speedup (520 ms) compared to the estimated prover time of LeOPaRd [ESTX26], making it the first practical, maliciously secure, post-quantum VOPRF. Moreover, the server verification time of 15 ms represents a $> 60\times$ improvement over LaBRADOR-style proofs, enabling servers to handle a large number of requests. While the performance looks promising, we note that there are still several aspects that can be further improved, which we leave as future work.

Instantiating the message mapping function H using SHAKE. In the current version of Beasley, we instantiate the message mapping function H as in lattice-based PRFs such as [BLMR13, BP14], and we use heuristic parameters which do not satisfy the noise bound required only in the PRF security proof. To bypass such proof artifacts, one can consider instantiating H with a secure hash function such as SHAKE. This would require the client to prove the correct evaluation of SHAKE. We expect the resulting running time to remain practical.

Further improving Akita’s proof time. Our current running time is dominated by the Akita PCS. We note that the current Akita implementation is mainly optimized for very large polynomials. There is still significant room for further improvement for small polynomials (below 2^{20} coefficients).

Making Akita zero-knowledge. Our implementation is not yet end-to-end zero-knowledge because Akita is not currently zero-knowledge. The authors of Akita have stated [DBK⁺26] that they plan to upgrade it to support zero-knowledge in the near future.

Parallelization and batching. Our implementation is currently single-threaded and processes only one VOPRF query at a time. However, we observe that our protocol could be efficient in the batch setting. In particular, the proof size and verification time are expected to remain nearly unchanged as the batch size increases. For instance, when batching 30 queries, we expect the NIZK proof size and verification time to remain below 80 KB and 50 ms, respectively. Moreover, both sumcheck and Akita are highly parallelizable, so Beasley could significantly benefit from multithreaded computation. With further optimization, we expect Beasley to support 30 VOPRF queries per second using four threads.

References

- [AAS25] Khutsaeva Altana, Leevik Anton, and Bezzateev Sergey. A survey of post-quantum oblivious protocols. *Cryptography*, 9(4):62, 2025.
- [ACFY25] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. Whir: Reed–solomon proximity testing with super-fast verification. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 214–243. Springer, 2025.
- [ADDS21] Martin R Albrecht, Alex Davidson, Amit Deo, and Nigel P Smart. Round-optimal verifiable oblivious pseudorandom functions from ideal lattices. In *IACR International Conference on Public-Key Cryptography*, pages 261–289. Springer, 2021.
- [AG24] Martin R Albrecht and Kamil Doruk Gur. Verifiable oblivious pseudorandom functions from lattices: Practical-ish and thresholdisable. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 205–237. Springer, 2024.
- [BBL⁺14] Abhishek Banerjee, Hai Brenner, Gaëtan Leurent, Chris Peikert, and Alon Rosen. Spring: Fast pseudorandom functions from rounded ring products. In *International Workshop on Fast Software Encryption*, pages 38–57. Springer, 2014.
- [BBL⁺26] Beatrice Biasioli, Madalina Bolboceanu, Vadim Lyubashevsky, Antonio Merino-Gallardo, Michał Osadnik, Gregor Seiler, and Patrick Steuer. A toolkit for succinct lattice-based zero knowledge proofs. *Cryptology ePrint Archive*, 2026. <https://eprint.iacr.org/2026/1289>.
- [BLMR13] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Annual Cryptology Conference*, pages 410–428. Springer, 2013.
- [BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In *Annual Cryptology Conference*, pages 353–370. Springer, 2014.
- [BS23] Ward Beullens and Gregor Seiler. Labrador: Compact proofs for r1cs from module-sis. In *Annual International Cryptology Conference*, pages 518–548. Springer, 2023.
- [Cre02] Giovanni Di Crescenzo. Equivocable and extractable commitment schemes. In *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 74–87. Springer, 2002.
- [DBK⁺26] Quang Dao, Omid Bodaghi, Amirhossein Khajepour, Giuseppe Vitto, Mohammadtaghi Badakhshan, Markos Georgiades, Fengrun Liu, Jiapeng Zhang, and Justin Thaler. Akita: A high-performance lattice-based polynomial commitment scheme. *Cryptology ePrint Archive*, 2026. <https://eprint.iacr.org/2026/1983>.

- [DDTT25] Alex Davidson, Amit Deo, and Louis Tremblay Thibault. Pool: A practical ot-based oprf from learning with rounding. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 1038–1052, 2025.
- [DGSW24] Alex Davidson, Sharon Goldberg, Nick Sullivan, and Christopher A. Wood. Oblivious pseudo-random functions (OPRFs) using prime-order groups. RFC 9497, December 2024.
- [ESTX26] Muhammed F Esgin, Ron Steinfeld, Erkan Tairi, and Jie Xu. Leopard: Towards practical post-quantum oblivious prfs via interactive lattice problems. *Annual Cryptology Conference*, 2026.
- [FLÖ25] Karla Friedrichs, Anja Lehmann, and Cavit Özbay. Game changer: A modular framework for OPRF security. In *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part VIII*, pages 582–613, 2025.
- [HKL⁺25] Lena Heimberger, Daniel Kales, Riccardo Lolato, Omid Mir, Sebastian Ramacher, and Christian Rechberger. Leap: A fast, lattice-based oprf with application to private set intersection. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 254–283. Springer, 2025.
- [HMZ26] Mi-Ying Miryam Huang, Xinyu Mao, and Jiapeng Zhang. Sublinear proofs over polynomial rings. *Asiacrypt*, 2026.
- [JKK14] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 233–253. Springer, 2014.
- [LSS24] Vadim Lyubashevsky, Gregor Seiler, and Patrick Steuer. The lazer library: Lattice-based zero knowledge and succinct proofs for quantum-safe privacy. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 3125–3137, 2024.
- [NOZ26] Ngoc Khanh Nguyen, George O’Rourke, and Jiapeng Zhang. Hachi: Efficient lattice-based multilinear polynomial commitments over extension fields. *Cryptology ePrint Archive*, 2026. <https://eprint.iacr.org/2026/156>.
- [Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.
- [Set20] Srinath Setty. Spartan: Efficient and general-purpose zk snarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.
- [TCR⁺22] Nirvan Tyagi, Sofia Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A. Wood. A fast and simple partially oblivious prf, with applications. In *EUROCRYPT (2)*, volume 13276 of *Lecture Notes in Computer Science*, pages 674–705. Springer, 2022.
- [Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2-4):117–660, 2022.
- [XZZ⁺19] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Annual International Cryptology Conference*, pages 733–764. Springer, 2019.
- [YBH⁺25] Yibin Yang, Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, and Tal Rabin. Gold oprf: Post-quantum oblivious power-residue prf. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 259–278. IEEE, 2025.

A Definitions of Verifiable OPRF

In this section, we present the formal definitions of a Verifiable Oblivious Pseudorandom Function (VOPRF), following the framework of [ESTX26, JKK14].

A.1 Syntax

A VOPRF is a two-party protocol between a client $\mathbb{C}$ holding a private input $x \in \mathcal{X}$ and a server $\mathbb{S}$ holding a secret key sk associated with a public key pk . At the end of the protocol execution, the client obtains the evaluation $y = F_{\text{sk}}(x) \in \mathcal{Y}$ without revealing x to the server, while convinced that the server evaluated the function correctly under the key committed in pk . At the same time, the server learns no information about the client's input x or the output y .

Formally, a VOPRF scheme $\mathcal{F}$ over an input space $\mathcal{X}$ and output space $\mathcal{Y}$ consists of the following algorithms:

- $\text{Setup}(1^\lambda) \rightarrow \text{pp}$: A randomized algorithm that, given the security parameter 1^λ , generates public parameters pp . We assume that pp is an implicit argument to the remaining algorithms.
- $\text{KeyGen}(\text{pp}) \rightarrow (\text{pk}, \text{sk})$: Given public parameters pp , it samples a master secret key sk and outputs a public key pk committing to sk .
- $\text{Request}(\text{pk}, x) \rightarrow (\text{st}_c, \text{req})$: Given public key pk and private input $x \in \mathcal{X}$, it computes an internal client state st_c and a evaluation request req .
- $\text{BlindEval}(\text{sk}, \text{pk}, \text{req}) \rightarrow \text{resp}$: Given secret key sk , public key pk , and a client request req , it verifies the request and outputs a blinded response resp , or $\perp$ if the request fails verification.
- $\text{Finalize}(\text{pk}, \text{st}_c, \text{req}, \text{resp}) \rightarrow y \in \mathcal{Y} \cup \{\perp\}$: Given public key pk , client state st_c , request req , and response resp , it verifies and un.masks the blinded response, and outputs the PRF evaluation $y \in \mathcal{Y}$, or aborts with $\perp$ if verification fails.
- $\text{Eval}(\text{sk}, x) \rightarrow y \in \mathcal{Y}$: The deterministic evaluation algorithm takes secret key sk and input $x \in \mathcal{X}$, and computes the PRF output $y = F_{\text{sk}}(x) \in \mathcal{Y}$.

A.2 Correctness

Correctness requires that an honest execution of the oblivious protocol between client and server yields the same value as desired PRF output, except with negligible probability.

Definition A.1 (Correctness [ESTX26, AG24]). A VOPRF scheme $\mathcal{F} = (\text{Setup}, \text{KeyGen}, \text{Request}, \text{BlindEval}, \text{Finalize}, \text{Eval})$ is correct for a correctness error $\epsilon(\lambda)$ if for all security parameters $\lambda \in \mathbb{N}$, all $\text{pp} \leftarrow \text{Setup}(1^\lambda)$, all $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$, and every input $x \in \mathcal{X}$:

$$\Pr \left\{ y = y' \mid \begin{array}{l} (\text{st}_c, \text{req}) \leftarrow \text{Request}(\text{pk}, x), \\ \text{resp} \leftarrow \text{BlindEval}(\text{sk}, \text{pk}, \text{req}), \\ y \leftarrow \text{Finalize}(\text{pk}, \text{st}_c, \text{req}, \text{resp}), \\ y' = \text{Eval}(\text{sk}, x) \end{array} \right\} \geq 1 - \text{negl}(\lambda).$$

A.3 Security Properties

Following [ESTX26, TCR⁺22, JKK14], we define in the following *pseudorandomness* against malicious clients and *request privacy* against malicious servers.

Pseudorandomness. Pseudorandomness (server security) guarantees that no efficient adversary corrupting the client can distinguish the PRF outputs on unqueried inputs from uniformly random strings, even after adaptively querying the server. This is formalized using a simulation-based definition.

Definition A.2 (Pseudorandomness [ESTX26, TCR⁺22]). Let $\mathcal{F}$ be a VOPRF scheme, let RO denote the set of random oracles, and let $\text{Func}(\mathcal{X} \rightarrow \mathcal{Y})$ be the set of functions from $\mathcal{X}$ to $\mathcal{Y}$. For a PPT adversary $\mathcal{A}$ and a PPT simulator $\mathcal{S} = (\mathcal{S}.\text{Setup}, \mathcal{S}.\text{BlindEval}, \mathcal{S}.\text{RO})$, we define the following experiment $\text{Expt}_{\mathcal{F}, \mathcal{A}, \mathcal{S}, \text{RO}}^{\text{prnd}-b}(1^\lambda)$ for $b \in \{0, 1\}$.

1. Setup Phase:

- If $b = 0$ (Ideal): $(\text{pp}, \text{pk}, \text{st}_{\mathcal{S}}) \leftarrow \mathcal{S}.\text{Setup}(1^\lambda)$, $F \leftarrow \text{Func}(\mathcal{X} \rightarrow \mathcal{Y})$.
- If $b = 1$ (Real): $\text{pp} \leftarrow \text{Setup}(1^\lambda)$, $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$, $\text{st}_{\text{RO}} \leftarrow \text{RO}.\text{Init}(1^\lambda)$.

2. Query Phase: The adversary $\mathcal{A}(\text{pp}, \text{pk})$ may adaptively query the following oracles:

- $\text{BlindEval}(\text{req})$: If $b = 0$, compute $\text{resp} \leftarrow \mathcal{S}.\text{BlindEval}^F(\text{req}, \text{st}_{\mathcal{S}})$. If $b = 1$, return $\text{resp} \leftarrow \text{BlindEval}(\text{pp}, \text{sk}, \text{pk}, \text{req})$.
- $\text{Eval}(x)$: If $b = 0$, return $y = F(x)$. If $b = 1$, return $y \leftarrow \text{Eval}(\text{pp}, \text{sk}, x)$.
- $\text{RO}(m)$: If $b = 0$, return $\mathcal{S}.\text{RO}(m, \text{st}_{\mathcal{S}})$. If $b = 1$, return $\text{RO}.\text{Eval}(m, \text{st}_{\text{RO}})$.

3. Output Phase: $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$, which is the output of the experiment.

The advantage of $\mathcal{A}$ against the pseudorandomness of $\mathcal{F}$ with respect to simulator $\mathcal{S}$ is defined as:

$$\text{Adv}_{\mathcal{F}, \mathcal{A}, \mathcal{S}}^{\text{prnd}}(\lambda) \triangleq \left| \Pr\{\text{Expt}_{\mathcal{F}, \mathcal{A}, \mathcal{S}, \text{RO}}^{\text{prnd}-1}(1^\lambda) = 1\} - \Pr\{\text{Expt}_{\mathcal{F}, \mathcal{A}, \mathcal{S}, \text{RO}}^{\text{prnd}-0}(1^\lambda) = 1\} \right|.$$

We say that a VOPRF scheme $\mathcal{F}$ has *pseudorandomness* if for every PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that $\text{Adv}_{\mathcal{F}, \mathcal{A}, \mathcal{S}}^{\text{prnd}}(\lambda) \leq \text{negl}(\lambda)$.

Request Privacy. Request privacy (client security or obliviousness) ensures that a malicious server learns nothing about the client's input x .

Definition A.3 (Request Privacy [ESTX26, TCR⁺22]). Let $\mathcal{F}$ be a VOPRF scheme and let RO denote the set of random oracles. For a PPT adversary $\mathcal{A}$, consider the following experiment $\text{Expt}_{\mathcal{F}, \mathcal{A}, \text{RO}}^{\text{req-priv}-b}(1^\lambda)$ for $b \in \{0, 1\}$.

1. Setup Phase: The challenger samples $\text{pp} \leftarrow \text{Setup}(1^\lambda)$, sets $D_r := \emptyset$, $q_r := 0$, initializes $\text{st}_{\text{RO}} \leftarrow \text{RO}.\text{Init}(1^\lambda)$, and sends pp to $\mathcal{A}$.

2. Query Phase: The adversary $\mathcal{A}$ may adaptively query the following oracles:

- $\text{Request}(\text{pk}, x_0, x_1)$: On input an arbitrary pk and challenge inputs $x_0, x_1 \in \mathcal{X}$:
 - (i) Compute $(\text{st}_{c,0}, \text{req}_0) \leftarrow \text{Request}(\text{pk}, x_0)$ and $(\text{st}_{c,1}, \text{req}_1) \leftarrow \text{Request}(\text{pk}, x_1)$;
 - (ii) If $\text{req}_0 = \perp$ or $\text{req}_1 = \perp$, return $\perp$;
 - (iii) Increment $q_r := q_r + 1$ and store $D_r[q_r] := (\text{pk}, \text{st}_{c,0}, \text{st}_{c,1}, \text{req}_0, \text{req}_1)$;
 - (iv) Return the pair $(\text{req}_b, \text{req}_{1-b})$ to $\mathcal{A}$.
- $\text{Finalize}(j, \text{resp}_0, \text{resp}_1)$: On input an index $j \in [1, q_r]$ and candidate server responses $(\text{resp}_0, \text{resp}_1)$:
 - (i) Retrieve $(\text{pk}, \text{st}_{c,0}, \text{st}_{c,1}, \text{req}_0, \text{req}_1) = D_r[j]$;
 - (ii) Compute $y_0 \leftarrow \text{Finalize}(\text{pk}, \text{st}_{c,b}, \text{req}_b, \text{resp}_0)$ and $y_1 \leftarrow \text{Finalize}(\text{pk}, \text{st}_{c,1-b}, \text{req}_{1-b}, \text{resp}_1)$;
 - (iii) If $y_0 = \perp$ or $y_1 = \perp$, return $\perp$;
 - (iv) Return (y_0, y_1) to $\mathcal{A}$.
- $\text{RO}(m)$: Return $\text{RO}.\text{Eval}(m, \text{st}_{\text{RO}})$.

3. **Output Phase:** $\mathcal{A}$ outputs a guess bit $b' \in \{0, 1\}$, which is the output of the experiment.

The advantage of $\mathcal{A}$ against the request privacy of $\mathcal{F}$ is defined as:

$$\text{Adv}_{\mathcal{F}, \mathcal{A}}^{\text{req-priv}}(\lambda) \triangleq \left| \Pr\{\text{Expt}_{\mathcal{F}, \mathcal{A}, \text{RO}}^{\text{req-priv-1}}(1^\lambda) = 1\} - \Pr\{\text{Expt}_{\mathcal{F}, \mathcal{A}, \text{RO}}^{\text{req-priv-0}}(1^\lambda) = 1\} \right|.$$

We say that a VOPRF scheme $\mathcal{F}$ has *request privacy* if for every PPT adversary $\mathcal{A}$, $\text{Adv}_{\mathcal{F}, \mathcal{A}}^{\text{req-priv}}(\lambda) \leq \text{negl}(\lambda)$.

Uniqueness and Verifiability. In addition to pseudorandomness and request privacy, a verifiable OPRF guarantees *uniqueness* [ESTX26, TCR⁺22]: a malicious server cannot produce two valid responses for the same input x and public key pk that finalize to different PRF values $y \neq y'$. As shown in [ESTX26, TCR⁺22], uniqueness is achieved if the commitment scheme COM is computationally key-binding and the protocol satisfies correctness and request privacy

B Additional Preliminaries

Uniformity of Message Mappings. Following [ESTX26], the correctness and privacy of the VOPRF rely on the statistical uniformity of the message mapping H .

Definition B.1 (ϵ_u -Uniformity [ESTX26]). A message mapping $H : \{0, 1\}^n \rightarrow \mathcal{R}_q^{h \times m}$ is ϵ_u -uniform if for any input $x \in \{0, 1\}^n$, the statistical distance between $H(x) \pmod{q}$ and the uniform distribution over $\mathcal{R}_q^{h \times m}$ is at most $\epsilon_u = \text{negl}(\lambda)$.

Extractable Commitment Schemes. An extractable commitment scheme [Cre02, ESTX26] enables a party to commit to a message $\mathbf{m}$ while allowing a simulator holding a trapdoor to extract $\mathbf{m}$ from the commitment.

Definition B.2 (Extractable Commitment Scheme [Cre02, ESTX26]). An extractable commitment scheme COM over message space $\mathcal{M}$ consists of a tuple of PPT algorithms (Setup , Commit , Verify , $\mathcal{E}_1$, $\mathcal{E}_2$):

- $\text{Setup}(1^\lambda) \rightarrow \text{ck}$: Outputs a public commitment key ck .
- $\text{Commit}(\text{ck}, \mathbf{m}; \rho) \rightarrow \mathbf{c}$: Given commitment key ck , message $\mathbf{m} \in \mathcal{M}$, and opening randomness ρ , produces a commitment $\mathbf{c}$. When ρ is sampled internally, we simply write $\text{Commit}(\text{ck}, \mathbf{m})$.
- $\text{Verify}(\text{ck}, \mathbf{c}, \rho, \mathbf{m}) \rightarrow \{0, 1\}$: Outputs 1 if $\mathbf{c} = \text{Commit}(\text{ck}, \mathbf{m}; \rho)$ and ρ is a valid randomness, and 0 otherwise.
- $\mathcal{E}_1(1^\lambda) \rightarrow (\text{ck}, \text{td})$: Randomized trapdoor setup algorithm that outputs a simulated commitment key ck along with a trapdoor td .
- $\mathcal{E}_2(\text{td}, \mathbf{c}) \rightarrow \mathbf{m} \in \mathcal{M} \cup \{\perp\}$: Given a trapdoor td , the extraction algorithm extracts the message $\mathbf{m}$ from commitment $\mathbf{c}$.

The scheme COM should satisfy:

1. **Correctness:** For all $\text{ck} \leftarrow \text{Setup}(1^\lambda)$, $\mathbf{m} \in \mathcal{M}$, and valid randomness ρ , we have $\text{Verify}(\text{ck}, \text{Commit}(\text{ck}, \mathbf{m}; \rho), \rho, \mathbf{m}) = 1$.
2. **Computational Hiding:** For any efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists negligible $\epsilon(\lambda)$ such that

$$\left| \Pr \left\{ b' = b \mid \begin{array}{l} b \leftarrow \{0, 1\}, \\ \text{ck} \leftarrow \text{Setup}(1^\lambda), \\ (\mathbf{m}_0, \mathbf{m}_1, \text{st}) \leftarrow \mathcal{A}_1(\text{ck}), \\ \mathbf{c} \leftarrow \text{Commit}(\text{ck}, \mathbf{m}_b), \\ b' \leftarrow \mathcal{A}_2(\text{st}, \mathbf{c}) \end{array} \right\} - \frac{1}{2} \right| \leq \epsilon(\lambda).$$

3. **Computational Existential Binding:** For any efficient adversary $\mathcal{A}$, there exists negligible $\epsilon(\lambda)$ such that

$$\Pr \left\{ \begin{array}{l} \exists (\mathbf{m}, \tilde{\mathbf{m}}, \rho, \tilde{\rho}) : \\ \mathbf{m} \neq \tilde{\mathbf{m}} \\ \text{Verify}(\text{ck}, \mathbf{c}, \rho, \mathbf{m}) = 1 \\ \text{Verify}(\text{ck}, \mathbf{c}, \tilde{\rho}, \tilde{\mathbf{m}}) = 1 \end{array} \middle| \begin{array}{l} (\text{ck}, *) \leftarrow \mathcal{E}_1(1^\lambda) \\ \mathbf{c} \leftarrow \mathcal{A}(\text{ck}) \end{array} \right\} \leq \epsilon(\lambda).$$

4. **Indistinguishable Simulated Key:** The distribution of ck produced by $\mathcal{E}_1(1^\lambda)$ is computationally indistinguishable from that of $\text{Setup}(1^\lambda)$.
5. **Extractability:** For all $(\text{ck}, \text{td}) \leftarrow \mathcal{E}_1(1^\lambda)$ and every tuple $(\mathbf{c}, \rho, \mathbf{m})$ satisfying $\text{Verify}(\text{ck}, \mathbf{c}, \rho, \mathbf{m}) = 1$, it holds that $\mathcal{E}_2(\text{td}, \mathbf{c}) = \mathbf{m}$ with high probability.

Regev-Style Instantiation over Module Lattices. Following [ESTX26, Appendix F], we instantiate the extractable commitment scheme COM using Regev encryption [Reg09] over the cyclotomic ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$. This structure natively integrates with our sumcheck-based NIZK arithmetization and is implemented in the Beasley Rust library.

Let n denote the commitment rank, $M \geq 1$ the message dimension in ring elements, and $w \geq 1$ a slack parameter providing hiding via the leftover hash lemma. The message $\mathbf{m} = (m_1, \dots, m_M)^\top \in \mathcal{R}_q^M$ has bounded coefficients $m_{i,j} \in \{0, \dots, b_m - 1\}$ for a small base $b_m \geq 2$ (in our implementation $b_m = 2$). The opening randomness $\rho \in \{-1, 0, 1\}^{(n+M+w)N}$ is ternary, represented in the NIZK witness as $\rho = \rho_{\text{pos}} - \rho_{\text{neg}}$ with binary vectors $\rho_{\text{pos}}, \rho_{\text{neg}} \in \{0, 1\}^{(n+M+w)N}$. The scheme proceeds as follows:

- **Setup**(1^λ): Sample uniformly random $\mathbf{A} \leftarrow \mathcal{R}_q^{n \times (n+M+w)}$ and $\mathbf{B} \leftarrow \mathcal{R}_q^{M \times (n+M+w)}$, and output commitment key $\text{ck} := (\mathbf{A}, \mathbf{B})$.
- $\mathcal{E}_1(1^\lambda)$: Sample $\mathbf{A} \leftarrow \mathcal{R}_q^{n \times (n+M+w)}$, secret matrix $\mathbf{S} \leftarrow \chi_s^{M \times n}$, and small error matrix $\mathbf{E} \leftarrow \chi_e^{M \times (n+M+w)}$. Set $\mathbf{B} := \mathbf{S}\mathbf{A} + \mathbf{E} \pmod{q} \in \mathcal{R}_q^{M \times (n+M+w)}$. Output $\text{ck} := (\mathbf{A}, \mathbf{B})$ and trapdoor $\text{td} := \mathbf{S}$. Key indistinguishability from uniform follows directly from $\text{MLWE}_{n,M,\mathcal{R}_q,\chi_s,\chi_e}$.
- **Commit**($\text{ck}, \mathbf{m}; \rho$): Given $\text{ck} = (\mathbf{A}, \mathbf{B})$, message $\mathbf{m} \in \mathcal{R}_q^M$, and short randomness $\rho \in \{-1, 0, 1\}^{(n+M+w)N}$, compute:

$$\begin{aligned} \mathbf{c}_0 &= \mathbf{A}\rho \pmod{q} \in \mathcal{R}_q^n, \\ \mathbf{c}_1 &= b_m \mathbf{B}\rho + \mathbf{m} \pmod{q} \in \mathcal{R}_q^M. \end{aligned}$$

Output commitment $\mathbf{c} := (\mathbf{c}_0, \mathbf{c}_1) \in \mathcal{R}_q^{n+M}$ and opening ρ .

- $\mathcal{E}_2(\text{td}, \mathbf{c})$: Given trapdoor $\text{td} = \mathbf{S}$ and commitment $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$, compute:

$$\mathbf{u} := \mathbf{c}_1 - b_m \mathbf{S}\mathbf{c}_0 \pmod{q} = b_m \mathbf{E}\rho + \mathbf{m} \pmod{q}.$$

Since $\|\mathbf{E}\rho\|_\infty < q/(2b_m)$, reducing each coefficient modulo b_m cancels the $b_m \mathbf{E}\rho$ term and exactly recovers $\mathbf{m} = \mathbf{u} \bmod b_m$.

- **Verify**($\text{ck}, \mathbf{c}, \rho, \mathbf{m}$): Parse $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1)$. Check that $\|\rho\|_\infty \leq 1$, $\mathbf{c}_0 = \mathbf{A}\rho \pmod{q}$, and $\mathbf{c}_1 = b_m \mathbf{B}\rho + \mathbf{m} \pmod{q}$.

Implementation in Beasley. In our implementation, we use three instances of COM with commitment keys $\text{ck}_1, \text{ck}_2, \text{ck}_3$:

1. $\mathbf{c}_r = \text{COM.Commit}(\text{ck}_3, \mathbf{R}; \rho_r)$: Commits to the client's masking randomness $\mathbf{R} \in \mathcal{R}_q^{h \times (\ell+m)}$, where we set $h = 1, \ell = m = 5$.
2. $\mathbf{d}_x = \text{COM.Commit}(\text{ck}_1, x; \rho_x)$: Commits to the client input $x \in \{0, 1\}^n$.
3. $\mathbf{c}_k = \text{COM.Commit}(\text{ck}_2, \mathbf{k}; \rho_k)$: Commits to the server secret key $\mathbf{k} \in \mathcal{R}_q^m$.

C Instantiation using BP14 PRF

We can also instantiate the LeOPaRd protocol using the BP14 PRF with some tweaks to reduce online latency and make the proof systems more efficient. In particular, we can apply our optimization for the BLMR PRF to BP14 PRF by using 2^w public random matrices to define the message mapping $H^{\text{bp}}(\cdot)$ for $w \geq 1$, where $w = 1$ corresponds to the original BP14 definition. Tuning the parameter w provides an interesting optimization opportunity for the online latency and the size of the required NIZK proof, at the cost of potentially larger memory footprint. Below we present such modified PRF and its security property.

Right spine tree with bitstring labeled nodes. As in [BP14], we consider full binary trees T , which are either a leaf node or an ordered pair (L, R) of full binary trees. The size of the tree T is denoted as $|T|$, which counts the number of the leaves.

Definition C.1 (BP14+message mapping). Let $w, k \geq 1$ be integers, and let $\{\mathbf{A}_{\mathbf{h}} \in \mathcal{R}_q^{m \times m\ell} \mid \mathbf{h} \in \{0, 1\}^w\}$ be a list of matrices over $\mathcal{R}_q$ labeled by bitstrings of length w . Given a bitstring $\mathbf{x} = \mathbf{x}_0 \cdots \mathbf{x}_{k-1} \in \{0, 1\}^{wk}$ as a concatenation of length w blocks $\mathbf{x}_i \in \{0, 1\}^w$ for all $i \in [k]$, we define the BP14+message mapping $H^{\text{bp}} : \{0, 1\}^{wk} \rightarrow \mathcal{R}_q^{m \times m\ell}$ as

$$H^{\text{bp}}(\mathbf{x}) \triangleq \mathbf{A}_{\mathbf{x}_0} \cdot G^{-1} \left(\mathbf{A}_{\mathbf{x}_1} \cdot G^{-1} \left(\cdots \mathbf{A}_{\mathbf{x}_{k-2}} \cdot G^{-1}(\mathbf{A}_{\mathbf{x}_{k-1}}) \cdots \right) \right). \quad (9)$$

In the OPRF protocol, we use a truncated message mapping $\bar{H}^{\text{bp}} : \{0, 1\}^{kw} \rightarrow \mathcal{R}_q^{h \times m\ell}$ where $\bar{H}^{\text{bp}}(\mathbf{x}) = H^{\text{bp}}(\mathbf{x})[1 : h]$ consisting of the first h rows of $H^{\text{bp}}(\mathbf{x})$.

The above message mapping essentially defines a “right spine” tree $T^{(k-1)} = (\text{leaf}_0, T^{(k-2)})$ with leaves $\text{leaf}_0, \dots, \text{leaf}_{k-1}$, where each leaf_i corresponds to a pre-defined public matrix $\mathbf{A}_{\mathbf{x}_i}$ indexed by the i 'th input block $\mathbf{x}_i \in \{0, 1\}^w$. Note that $e(T^{(k)}) = 1$ for all $k \geq 1$.

Definition C.2 (BP14+PRF). Let $w, k \geq 1$ be integers, and let $\mathbb{A} = \{\mathbf{A}_{\mathbf{h}} \in \mathcal{R}_q^{m \times m\ell} \mid \mathbf{h} \in \{0, 1\}^w\}$ be matrices over $\mathcal{R}_q$. Define the function family, parameterized by matrices in $\mathbb{A}$ and a modulus $p < q$:

$$\mathcal{F}_{\mathbb{A}, p} \triangleq \{F_{\mathbf{k}} : \{0, 1\}^{kw} \rightarrow \mathcal{R}_q^{m\ell} \mid \mathbf{k} \in \mathcal{R}_q^{m\ell}\},$$

where the functions $F_{\mathbf{k}}$ are defined as

$$F_{\mathbf{k}}(\mathbf{x}) \triangleq \lfloor \mathbf{k}^\top \cdot H^{\text{bp}}(\mathbf{x}) \rfloor_p.$$

We now adopt the [BP14] security theorem to our modified PRF instantiation.

Theorem C.3 (PRF security). Let χ_e be some distribution over $\mathcal{R}$ with subgaussian parameter $r > 0$, and let q, p, k, w be parameters satisfying

$$q \geq p \cdot r \sqrt{k} \cdot k\ell \cdot \lambda^{\omega(1)}.$$

Let $\mathbf{A}_{\mathbf{h}}$ for all $\mathbf{h} \in \{0, 1\}^w$ be independently and uniformly sampled from $\mathcal{R}_q^{m \times m\ell}$. Assume decisional-MLWE $_{d, q, m, \chi_s, \chi_e}$ is hard. Then $\mathcal{F}_{\mathbb{A}, p}$ with secret key $\mathbf{k}$ chosen from a distribution χ_s is a secure PRF family.

Proof (Sketch). The proof is inspired by [BP14]. The difference is that leaf nodes now correspond to $\mathbf{A}_{\mathbf{h}}$ for some $\mathbf{h} \in \{0, 1\}^w$. Fix $x \in \{0, 1\}^{kw}$, and for all $0 \leq i \leq k-1$, define

$$\mathbf{A}_{x^{(i)}} \triangleq G^{-1}(\mathbf{A}_{x_{i+1}} \cdot G^{-1}(\cdots (\mathbf{A}_{x_{k-2}} \cdot G^{-1}(x_{k-1})) \cdots)) \in \mathcal{R}_q^{m \times m\ell}.$$

Then we can show that, under the LWE assumption with noise distribution χ_s , the function $G_{\mathbf{k}}(x) = \mathbf{k}^\top \cdot H^{\text{bp}}(x) + e(x)$ is computationally indistinguishable from a random function, where $e(x)$ is

$$e(x) = \sum_{i=0}^{k-2} e_i^\top \cdot \mathbf{A}_{x^{(i)}} + e_{k-1}^\top, \text{ for } e_i \leftarrow \chi_s.$$

With high probability we have $\|e(x)\|_\infty \leq k\ell \cdot r \sqrt{k}$. So, when $q/p > \lambda^{\omega(1)} \cdot \|e(x)\|_\infty$, we have that $\lfloor \mathbf{k}^\top \cdot H^{\text{bp}}(x) \rfloor_p$ is statistically indistinguishable from $\lfloor G_{\mathbf{k}}(x) \rfloor_p$. Therefore $\mathcal{F}_{\mathbb{A}, p}$ is a secure PRF family as claimed. $\square$