LNRPIR: High Throughput PIR with Flexible Communication-Computation Trade-offs

Liren Xu[1] and Zhe Li[2]

[1] School of Cyber Engineering, Xidian University, Xi'an, China
24151111259@stu.xidian.edu.cn
[2] Institute for Mathematics and Interdisciplinary Sciences, Xidian University, Xi'an, China
lizh0048@e.ntu.edu.sg

Abstract. Private information retrieval (PIR) is a powerful cryptographic primitive with numerous applications. Recent works have largely pursued two different directions: minimizing online communication at the expense of high online computation costs, or achieving high server throughput while incurring substantial online communication overhead. In this work, we introduce LNRPIR, a PIR scheme that achieves high server throughput with moderate online communication overhead. LNRPIR also provides flexible trade-offs between communication and computation, addressing a gap in the trade-off space left by prior works. Our evaluation shows that LNRPIR retrieves a 256-byte record from an 8 GB database 2.541× faster than VIA-C (S&P 2026), a state-of-the-art protocol in the client-hint model, with only 10.473 KB of online communication. Furthermore, compared with InsPIRe (S&P 2026), a state-of-the-art protocol in the silent-preprocessing model, LNRPIR achieves 17.074× lower online communication overhead for an 8 GB database, while incurring only 1.885× the online computation time of InsPIRe. LNRPIR primarily relies on three novel ciphertext conversion techniques: LWE-to-NTRU conversion, LWE-to-VNTRU conversion, and NTRU-to-RLWE conversion.

1 Introduction

Suppose a client wants to retrieve a single entry from a public database while hiding which entry is requested from the server hosting the database. Cryptographic protocols that achieve this goal are known as private information retrieval (PIR) protocols. Since its introduction by Chor et al. [CGKS95], PIR has been extensively studied and developed. This powerful cryptographic primitive has found applications in a wide range of areas, including private web search [HDCZ23], private DNS lookup [NPVW27], anonymous communication [MOT+11], and certificate transparency [HHC+23], among others.

PIR protocols have been constructed in both multi-server and single-server settings. In the multi-server setting, the database is replicated across multiple servers, and the privacy guarantee relies on the assumption that the servers

do not collude. In the single-server setting, modest cryptographic assumptions are used in place of the non-collusion assumption. In the early stages of research, number-theoretic assumptions were commonly employed. Although some recent works still rely on these assumptions and achieve notably low communication costs, their throughput remains significantly limited (e.g., [ALP+21]). More recently, lattice-based assumptions have been used to establish new trade-offs between communication and computation. These assumptions have been extensively studied, and lattice-based schemes are well developed. Some schemes enable the server to send a hint to the client before queries are made, thereby reducing online computation. Others allow the client to send encrypted key material to the server in advance, enabling complex homomorphic computation. Recently, several schemes have been proposed that eliminate offline communication (see Section 6 for more information and references). In this paper, we focus on the single-server setting. Specifically, we propose a PIR protocol in which the server stores client-specific keys. This setting is also known as the "client-hint" model.

Performance goals of PIR protocols. When evaluating the performance of PIR protocols, we consider computational overhead (or, equivalently, server throughput), online communication cost (including both query and response), and offline communication cost for protocols that require data transmission before queries are made.

Bridging the gap in the PIR trade-off space. For an 8 GB database, some schemes achieve server throughput on the order of $1 \sim 10$ GB/s, but incur online communication costs on the order of hundreds of KB or more (e.g., SimplePIR [HHC+23] and InsPIRe [MPSY26]). Others reduce online communication costs to the order of 1 KB, but achieve server throughput only on the order of hundreds of MB/s (e.g., VIA-C [LWZ26]).

However, existing protocols do not offer a combination of relatively high server throughput and moderate online communication cost. Our goal is to bridge this gap by achieving server throughput on the order of 1 GB/s while keeping online communication cost on the order of 10 KB, thereby providing more flexible trade-offs between communication and computation. See Figure 9 for an intuitive illustration.

The LNRPIR protocol. The LNRPIR protocol achieves the aforementioned trade-offs by leveraging the core ideas of LWE-to-NTRU conversion (as well as LWE-to-VNTRU conversion) and NTRU-to-RLWE conversion. For an 8 GB database, LNRPIR achieves a server throughput of 1540.718 MB/s, which is 2.541× higher than that of VIA-C and 3.355× higher than that of Respire. Furthermore, under an adjusted parameter configuration, LNRPIR achieves a server throughput of 3186.309 MB/s, only 1.885× lower than InsPIRe, while the online communication cost of LNRPIR is 36.780 KB, which is 17.074× lower than that of InsPIRe. Overall, our protocol provides flexible trade-offs between communication and computation, thereby filling in the aforementioned gap.

An example application. Here we provide an example application for our protocol. Private DNS lookup [NPVW27] is a scenario in which queries are

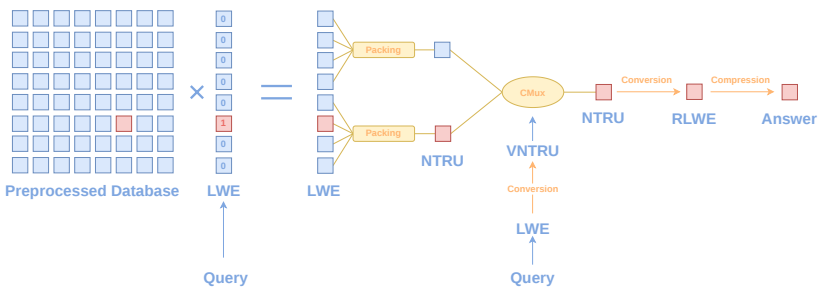

Fig. 1. An illustration of the main routine of LNRPIR.

made frequently, making high server throughput essential. Meanwhile, individual users may also generate many queries, making low online communication cost desirable. Moreover, the data transmitted offline in our protocol is independent of the database, so there is no need to be concerned about database updates. Therefore, our protocol is suitable for implementing private DNS lookup.

1.1 Our Techniques

Main routine of homomorphic computation. In many recent PIR protocols, the database is viewed as a matrix of records (e.g., InsPIRe [MPSY26], VIA-C [LWZ26]), and homomorphic selections are performed for both dimensions of the matrix. In LNRPIR, we follow the same idea. In the first dimension, we multiply the database by LWE encryptions. This brings two benefits: first, the online part of this multiplication is efficient; second, a query consisting of LWE encryptions has a remarkably small size when the random components are replaced by a PRG seed.

Then we pack the resulting LWE encryptions into NTRU encryptions. This packing technique also brings two benefits. On the one hand, we manage to realize an efficient packing algorithm. This is important since the packing step is typically time-consuming, not to mention that in our routine we perform packing for numerous groups of LWE encryptions. On the other hand, NTRU encryptions also enable the homomorphic CMux operation, which is a 1-out-of-2 selection with only additive error growth. Moreover, NTRU-based CMux is naturally faster than RLWE-based CMux.

Next, for the second dimension, we perform several rounds of CMux to obtain only one NTRU encryption. Using additional techniques, it can be further converted into an RLWE encryption and then compressed to reduce the response size. Figure 1 illustrates a toy example of the main routine.

LWE-to-NTRU conversion. Following the ideas behind LWE-to-RLWE conversion or ring packing employed in many recent works (e.g., YPIR [MW24], InsPIRe [MPSY26], VIA-C [LWZ26]), we propose the LWE-to-NTRU conversion with a special optimization that is not realized in the RLWE case. Specifically, we convert one LWE encryption into one NTRU encryption by performing homomorphic decryption of the LWE encryption, while the dimension and modulus

of the LWE encryption can, at the same time, be much smaller than those of the NTRU encryption, respectively. This optimization has two advantages. First, a smaller modulus of LWE encryptions contributes to a smaller query size. Second, a smaller modulus of LWE encryptions also speeds up the multiplication between these LWEs and the database, which is one of the most time-consuming steps of the protocol. Furthermore, the LWE-to-NTRU conversion itself also incurs relatively little online computation, since it consists merely of a linear combination of switching keys for conversion.

Packing LWEs into one NTRU. As mentioned above, we actually want to pack multiple LWE encryptions into one NTRU encryption (similar to what is done in InsPIRe [MPSY26]), which is easy with the LWE-to-RLWE conversion. We only need to convert each LWE into NTRU, then rotate and add them. The rotation can be optimized by multiplying the switching keys by rotation monomials in advance; hence, the extra computation consists merely of adding polynomials.

LWE-to-VNTRU conversion. In order to realize the homomorphic CMux operation, we need an NTRU ciphertext multiplication operation similar to the RLWE-RGSW external product. This operation can be realized by leveraging GSW-like NTRU encryptions called vector NTRU (VNTRU) encryptions [WXZ+25]. Interestingly, we can also convert LWE encryptions into one VNTRU encryption, using similar ideas from the LWE-to-NTRU conversion. The only difference is that the LWEs and the VNTRU must have the same modulus here. Therefore, the query of our protocol only consists of LWE encryptions.

NTRU-based CMux. Now, with NTRU and VNTRU encryptions, we can perform the NTRU-VNTRU external product in exactly the same way as the RLWE-RGSW external product. Thus, we can perform CMux for the selection of the second dimension in our protocol. The inputs to the CMux gate are two NTRU encryptions packed from LWE encryptions resulting from the first-dimension selection, and a VNTRU encryption converted from LWE encryptions in the query. This NTRU-based CMux has an additional benefit. One can see that, given VNTRU and RGSW encryptions with the same gadget dimension, the NTRU-VNTRU external product is about $4\times$ faster than the RLWE-RGSW external product. Therefore, we also achieve an acceleration in the homomorphic computation of CMux, which is another time-consuming step.

NTRU-to-RLWE conversion. In order to reduce the response size, Respire [BMW24] introduced the response compression technique for RLWE encryptions. However, the extraction method used in this technique is incompatible with NTRU encryptions because of their fractional form. Thus we introduce the NTRU-to-RLWE conversion to help us convert an NTRU response into an RLWE response, which can then be compressed in a similar way to that in Respire.

2 Preliminaries

2.1 Notation

Let λ be the security parameter. For an integer $n \in \mathbb{Z}_{>0}$, let $[n] := \{0, 1, \ldots, n-1\}$. All vectors, unless otherwise stated, are column vectors. For a vector $\mathbf{a}$, let $\mathbf{a}_i$ denote the i-th element of $\mathbf{a}$. For an $m \times n$ matrix $\mathbf{A}$, let $\mathbf{A}_{i,j}$ denote the entry in the i-th row and j-th column, where $i \in [m], j \in [n]$. For integers $x, y \in \mathbb{Z}_{>0}$, let $x \mid y$ denote that x divides y. Vectors and matrices are denoted by bold lowercase letters and bold uppercase letters, respectively. For a distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ means to sample x from $\mathcal{D}$. For a finite set S, $x \xleftarrow{R} S$ means to draw a uniform random sample from S. Let R_n and $R_{n,q}$ denote the ring $\mathbb{Z}[x]/(x^n+1)$ and $\mathbb{Z}_q[x]/(x^n+1)$ respectively, where n is a power of 2.

2.2 Private Information Retrieval

As in [BMW24], we consider the single-server private information retrieval (PIR) protocol, where the client uploads reusable key material before making queries (also called the client hint), and the server performs preprocessing on the database before answering queries.

Definition 2.1 (Private Information Retrieval, [BMW24], adapted) *Let λ be the security parameter, $N = N(\lambda)$ be the number of records in the database, and $\mathcal{M} = \{\mathcal{M}_\lambda\}$ be the space of possible record values. Let* pp *be the set of public parameters given by λ, N, and $\mathcal{M}$. A single-server private information retrieval (PIR) protocol in the client-hint model is a tuple of efficient algorithms* $(\mathsf{DB.Setup}, \mathsf{Key.Gen}, \mathsf{Query.Gen}, \mathsf{Answer}, \mathsf{Recover})$ *with the following syntax:*

1. $\mathsf{DB.Setup}\left(\mathsf{pp}, \{\mathsf{d_i}\}_{i\in[N]}\right)$ *: On input the public parameters* pp *and a collection of records* $\mathsf{d_1}, \ldots, \mathsf{d_N} \in \mathcal{M}_\lambda$*, the database setup algorithm outputs a (preprocessed) database* db*.*
2. $\mathsf{Key.Gen}\,(\mathsf{pp})$*: On input the public parameters* pp*, the key generation algorithm outputs a query key* qk *and a client hint* H*.*
3. $\mathsf{Query.Gen}\,(\mathsf{pp}, \mathsf{idx}, \mathsf{qk})$*: On input the public parameters* pp*, an index* idx *and the query key* qk*, the query generation algorithm outputs a query* qu*.*
4. $\mathsf{Answer}\,(\mathsf{pp}, \mathsf{db}, \mathsf{H}, \mathsf{qu})$*: On input the public parameters* pp*, a preprocessed database* db*, the client hint* H*, the query* qu*, the* Answer *algorithm outputs an answer* res*.*
5. $\mathsf{Recover}\,(\mathsf{qk}, \mathsf{res})$*: On input the query key* qk *and the answer* res*, the* $\mathsf{Recover}$ *algorithm outputs a record* $\mathsf{d_i}$*.*

The algorithms must satisfy the following properties:

1. $\mathsf{Correctness}$*: For all security parameters $\lambda \in \mathbb{N}$, all sets of records $\mathsf{d_1}, \ldots, \mathsf{d_N} \in \mathcal{M}_\lambda$, and all indices $\mathsf{idx} \in [N]$, if*

$$\mathsf{db} \leftarrow \mathsf{DB.Setup}\left(\mathsf{pp}, \{\mathsf{d_i}\}_{i\in[N]}\right), (\mathsf{qk}, \mathsf{H}) \leftarrow \mathsf{Key.Gen}\,(\mathsf{pp}),$$

$$\mathsf{qu} \leftarrow \mathsf{Query.Gen}\left(\mathsf{pp}, \mathsf{idx}, \mathsf{qk}\right), \mathsf{res} \leftarrow \mathsf{Answer}\left(\mathsf{pp}, \mathsf{db}, \mathsf{H}, \mathsf{qu}\right),$$

then $\Pr\left[\mathsf{Recover}\left(\mathsf{qk}, \mathsf{res}\right) = \mathsf{d}_{\mathsf{idx}}\right] \geq 1 - c$. *We refer to* c *as the correctness error.*

2. Query privacy*: For a bit* $b \in \{0,1\}$ *and an adversary* $\mathcal{A}$*, we define the query privacy game as follows:*
   (a) *The challenger starts by sampling* $(\mathsf{qk}, \mathsf{H}) \leftarrow \mathsf{Key.Gen}\left(\mathsf{pp}\right)$ *and gives* H *to* $\mathcal{A}$ *(assuming* $\mathcal{A}$ *knows the public parameters and the database).*
   (b) $\mathcal{A}$ *can now make (arbitrarily many) queries on pairs of indices* $(\mathsf{idx}_0, \mathsf{idx}_1)$ *where* $\mathsf{idx}_0, \mathsf{idx}_1 \in [N]$*. On each query, the challenger responds with* $\mathsf{Query.Gen}\left(\mathsf{pp}, \mathsf{idx}_b, \mathsf{qk}\right)$*.*
   (c) *When* $\mathcal{A}$ *is done making queries, it outputs a bit* $b' \in \{0,1\}$*, which is the output of the experiment.*

   We say that the scheme satisfies query privacy if for all efficient adversaries $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\left|\Pr\left[b' = 1 \mid b = 0\right] - \Pr\left[b' = 1 \mid b = 1\right]\right| = \mathsf{negl}(\lambda).$$

2.3 Lattice Basics

Subgaussian random variables [Pei16]. A random variable X is subgaussian with width (or parameter) σ if $\Pr[|X| > t] \leq 2\exp\left(-\pi t^2/\sigma^2\right)$ for all $t \geq 0$. A discrete Gaussian variable with width σ is subgaussian with width σ. If X is subgaussian with width σ, cX is subgaussian with width $|c|\sigma$. The sum $X_1 + \ldots + X_k$ of independent subgaussian variables with widths $\sigma_1, \ldots, \sigma_k$ is subgaussian with width $\sqrt{\sum_i \sigma_i^2}$.

Rounding. Let $\lfloor \cdot \rceil : \mathbb{R} \to \mathbb{Z}$ denote the function that rounds the input to the nearest integer. For positive integers $q > p$, let $\lfloor \cdot \rceil_{q,p} : \mathbb{Z}_q \to \mathbb{Z}_p$ denote the function that takes as input $x \in \mathbb{Z}_q$, lifts it to an integer $x' \in (-q/2, q/2]$, and outputs $\lfloor p/q \cdot x' \rceil$. It is easy to extend these functions to inputs in R_n and $R_{n,q}$ by evaluating the coefficients separately.

Learning with Errors [Reg05]. Let $n = n(\lambda)$ be the dimension of the secret, $m = m(\lambda)$ be the number of samples, $q = q(\lambda)$ be the modulus, and $\chi_s = \chi_s(\lambda), \chi_e = \chi_e(\lambda)$ be distributions over $\mathbb{Z}_q$. The learning with errors (LWE) assumption $\mathsf{LWE}_{n,m,q,\chi_s,\chi_e}$ states that the following distributions are computationally indistinguishable:

$$\left\{(\mathbf{A}, \mathbf{A}\cdot\mathbf{s} + \mathbf{e}) : \mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{m\times n}, \mathbf{s} \leftarrow \chi_s^n, \mathbf{e} \leftarrow \chi_e^m\right\} \quad \text{and} \quad \left\{(\mathbf{A}, \mathbf{u}) : \mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{m\times n}, \mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m\right\}.$$

An LWE encryption (or ciphertext) of a message $m \in \mathbb{Z}_p$ is a vector $(\mathbf{a}, b) := (\mathbf{a}, \mathbf{a}\cdot\mathbf{s} + \Delta\cdot m + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ where $\mathbf{a} \xleftarrow{R} \mathbb{Z}_q^n$ is a uniformly random vector, $\mathbf{s} \leftarrow \chi_s^n$ is the secret key, $e \leftarrow \chi_e$ is an error term, and $\Delta = \lfloor q/p \rfloor$ is a scaling factor. When the error is small ($|e| < \frac{q}{2p} - (q \bmod p)$), the message can be recovered given $\mathbf{s}$. Specifically, $m = \lfloor b - \mathbf{a}\cdot\mathbf{s} \rceil_{q,p}$.

Ring Learning with Errors [LPR10]. Let $R_n = \mathbb{Z}[x]/(x^n + 1)$ where $n = n(\lambda)$ is a power of two. Let $m = m(\lambda)$ be the number of samples, $q = q(\lambda)$ be the modulus, and $\chi_s = \chi_s(\lambda), \chi_e = \chi_e(\lambda)$ be distributions over $R_{n,q} =$

$\mathbb{Z}_q[x]/(x^n+1)$. The ring learning with errors (RLWE) assumption $\mathsf{RLWE}_{n,m,q,\chi_s,\chi_e}$ states that the following distributions are computationally indistinguishable:

$$\left\{(\mathbf{a}, s\mathbf{a}+\mathbf{e}) : \mathbf{a} \xleftarrow{R} R_{n,q}^m, s \leftarrow \chi_s, \mathbf{e} \leftarrow \chi_e^m\right\} \quad \text{and} \quad \left\{(\mathbf{a}, \mathbf{u}) : \mathbf{a}, \mathbf{u} \xleftarrow{R} R_{n,q}^m\right\}.$$

An RLWE encryption of a message $m \in R_{n,p}$ is a pair $(a,b)^\top := (a, a\cdot s + \Delta\cdot m + e)^\top \in R_{n,q}^2$ where $a \xleftarrow{R} R_{n,q}$ is a uniformly random polynomial, $s \leftarrow \chi_s$ is the secret key, $e \leftarrow \chi_e$ is an error polynomial, and $\Delta = \lfloor q/p \rfloor$ is a scaling factor. When the error is small ($\|e\|_\infty < \frac{q}{2p} - (q \bmod p)$), the message can be recovered given s. Specifically, $m = \lfloor b - a\cdot s \rceil_{q,p}$.

NTRU [KF17, XZD+23]. Let $R_n = \mathbb{Z}[x]/(x^n+1)$ where $n = n(\lambda)$ is a power of two. Let $m = m(\lambda)$ be the number of samples, $q = q(\lambda)$ be the modulus, and $\chi_f = \chi_f(\lambda), \chi_g = \chi_g(\lambda)$ be distributions over $R_{n,q} = \mathbb{Z}_q[x]/(x^n+1)$. The NTRU assumption $\mathsf{NTRU}_{n,m,q,\chi_f,\chi_g}$ states that the following distributions are computationally indistinguishable:

$$\left\{(\frac{g_0}{f}, \ldots, \frac{g_{m-1}}{f}) : f \leftarrow \chi_f, g_0, \ldots, g_{m-1} \leftarrow \chi_g\right\} \quad \text{and} \quad \left\{\mathbf{u} : \mathbf{u} \xleftarrow{R} R_{n,q}^{1\times m}\right\}.$$

A (scalar) NTRU encryption of a message $m \in R_{n,p}$ is a polynomial $\mathsf{ct} := \frac{g+\Delta\cdot m}{f} \in R_{n,q}$ where $f \leftarrow \chi_f$ is the secret key invertible in $R_{n,q}$, $g \leftarrow \chi_g$ is an error polynomial, and $\Delta = \lfloor q/p \rfloor$ is a scaling factor. When the error is small ($\|g\|_\infty < \frac{q}{2p} - (q \bmod p)$), the message can be recovered given f. Specifically, $m = \lfloor \mathsf{ct}\cdot f \rceil_{q,p}$.

A vector NTRU (VNTRU) encryption of a message $m \in R_{n,p}$ is a vector $(\frac{g_0}{f} + B^0 m, \ldots, \frac{g_{d-1}}{f} + B^{d-1}m)^\top \in R_{n,q}^d$ where $f \leftarrow \chi_f$ is the secret key invertible in $R_{n,q}$, $g_0, \ldots, g_{d-1} \leftarrow \chi_g$ are error polynomials, and d is the dimension with respect to modulus q and the gadget base B, as described below.

Gadget vector and gadget decomposition [MP12]. For a modulus $q \in \mathbb{N}$ and a decomposition base $z \in \mathbb{N}$, the gadget vector is $\mathbf{g}_z := (1, z, z^2, \ldots, z^{\ell-1})^\top \in \mathbb{Z}_q^\ell$, where $\ell = \lfloor \log_z q \rfloor + 1$ is called the dimension. Let $\mathbf{g}_z^{-1} : \mathbb{Z}_q \to \mathbb{Z}_q^\ell$ denote the base-z digit decomposition operator, whose output (row) vectors have every element in $(-z/2, z/2]$ (unless otherwise specified). It can be easily extended to operate on $R_{n,q}$ by applying the decomposition to each coefficient of a polynomial in $R_{n,q}$ independently, that is, $\mathbf{g}_z^{-1} : R_{n,q} \to R_{n,q}^\ell$.

External product [WXZ+25]. Consider a scalar NTRU encryption $c_1 := \frac{g_1+\Delta\mu}{f} \in R_{n,q}$ and a vector NTRU encryption $c_2 := (\frac{g_{2,0}}{f} + B^0\nu, \ldots, \frac{g_{2,d-1}}{f} + B^{d-1}\nu)^\top \in R_{n,q}^d$ under the same secret key f, where d is the dimension of the gadget vector with respect to modulus q and decomposition base B. The external product between c_1 and c_2 is:

$$c_1 \odot c_2 = g_B^{-1}(c_1)\cdot c_2 = \frac{g_B^{-1}(c_1)\cdot \mathbf{g_2} + \nu\cdot g_1 + \Delta\cdot\mu\cdot\nu}{f},$$

where $\mathbf{g_2} := (g_{2,0}, \ldots, g_{2,d-1})^\top$. The result is a scalar NTRU encryption of $\mu\cdot\nu$ with respect to the same secret key f. Its error term is $g = g_B^{-1}(c_1)\cdot\mathbf{g_2} + \nu\cdot g_1$.

Independence heuristic. As in [MPSY26, BMW24, MW22] and many other prior works, we assume the independence heuristic where the error terms of all intermediate encodings in homomorphic computation are independent. We thereby analyze the widths of the subgaussian errors.

3 Essential Techniques

Packing LWE encryptions. The idea of transforming LWE encryptions into a single polynomial-form encryption to facilitate further homomorphic computation is useful, and is also employed in previous protocols like YPIR [MW24] and InsPIRe [MPSY26] where LWEs are transformed into one RLWE. In our protocol, we transform LWE encryptions into one NTRU encryption or one VNTRU encryption.

Transforming one LWE encryption into one NTRU encryption. Given an LWE encryption $(\mathbf{a}, b) := (\mathbf{a}, \mathbf{a}\cdot\mathbf{s} + \Delta\cdot m + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, the goal is to obtain an NTRU encryption $\frac{g+\Delta\cdot m}{f} \in R_{n,q}$. We adopt homomorphic decryption to recover m with the help of switching keys. Specifically, given the LWE encryption mentioned above and additional switching keys

$$K^{(b)} := \frac{\hat{g}+1}{f}, K_i^{(a)} := \frac{g_i + \mathbf{s}_i}{f} \in R_{n,q} \quad (i \in [n]),$$

where f is the secret key of NTRU encryptions, $\hat{g}, g_0, \ldots, g_{n-1} \leftarrow \chi_g$ are error terms, compute:

$$b\cdot K^{(b)} - \sum_{i=0}^{n-1} \mathbf{a}_i \cdot K_i^{(a)} = \frac{b\cdot\hat{g} - \sum_{i=0}^{n-1} \mathbf{a}_i \cdot g_i + \Delta\cdot m + e}{f} \in R_{n,q}.$$

The result is an NTRU encryption of m with secret key f and error term $g = b\cdot\hat{g} - \sum_{i=0}^{n-1} \mathbf{a}_i \cdot g_i + e$.

Optimization for reducing error growth. We apply gadget decomposition to $\mathbf{a}_i$ $(i \in [n])$ to reduce the main term of the error of the NTRU encryption constructed above. To be specific, let $\mathbf{g}_z := (1, z, z^2, \ldots, z^{\ell-1})^\top \in \mathbb{Z}_q^\ell$ be the gadget vector with respect to modulus q and base z. Now the switching keys are:

$$K^{(b)} := \frac{\hat{g}+1}{f}, K_{i,j}^{(a)} := \frac{g_{i,j} + z^j\cdot\mathbf{s}_i}{f} \in R_{n,q} \quad (i \in [n], j \in [\ell]),$$

where f is the secret key of NTRU encryptions, $\hat{g}, g_{i,j} \leftarrow \chi_g$ $(i \in [n], j \in [\ell])$ are error terms. Given the LWE encryption mentioned above, we first decompose $\mathbf{a}_i$ and obtain $\mathbf{a}_i = \sum_{j=0}^{\ell-1} a_{i,j}\cdot z^j$ where $a_{i,j} \in [z]$, then compute:

$$b\cdot K^{(b)} - \sum_{i=0}^{n-1}\sum_{j=0}^{\ell-1} a_{i,j}\cdot K_{i,j}^{(a)} = \frac{b\cdot\hat{g} - \sum_{i=0}^{n-1}\sum_{j=0}^{\ell-1} a_{i,j}\cdot g_{i,j} + \Delta\cdot m + e}{f} \in R_{n,q}.$$

The result is an NTRU encryption of m with secret key f and error term $g = b\cdot\hat{g} - \sum_{i=0}^{n-1}\sum_{j=0}^{\ell-1} a_{i,j}\cdot g_{i,j} + e$.

Remark. The gadget decomposition is NOT applied to b for the purpose of reducing online computation.

Further optimization for reducing the query size. As is shown later in the full protocol, the elements b of the LWE encryptions constitute the query. To reduce the query size, we adopt smaller lattice parameters for the LWE encryptions than those for NTRU encryptions. More precisely, let $(\mathbf{a}, b) := (\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + \delta \cdot m + e) \in \mathbb{Z}_{q_0}^{n_0} \times \mathbb{Z}_{q_0}$ be an LWE encryption, where the message $m \in \mathbb{Z}_p$ and $\delta = \lfloor q_0/p \rfloor$. Let $\mathbf{g}_z := (1, z, z^2, \ldots, z^{\ell-1})^\top \in \mathbb{Z}_q^\ell$ be the gadget vector with respect to modulus q_0 and base z. Without loss of generality, we assume $b \in [q_0], \mathbf{a}_i \in [q_0], \mathbf{s}_i \in (-q_0/2, q_0/2], i \in [n_0], m \in (-p/2, p/2]$ (viewed as elements in $\mathbb{Z}$). Let $\mathbf{a} \cdot \mathbf{s} + \delta \cdot m + e = b + \theta \cdot q_0$, $\theta \in \mathbb{Z}$. Now the switching keys are:

$$K^{(b)} := \frac{\hat{g} + \Delta}{f}, K_{i,j}^{(a)} := \frac{g_{i,j} + z^j \cdot \mathbf{s}_i \cdot \Delta}{f} \in R_{n_1,q_1} \quad (i \in [n], j \in [\ell]),$$

where $n_1 \geq n_0$, $q_1 \geq q_0$, $\Delta = \lfloor q_1/q_0 \rfloor$, f is the secret key of NTRU encryptions, $\hat{g}, g_{i,j} \leftarrow \chi_g$ $(i \in [n], j \in [\ell])$ are the error terms, and $\mathbf{s}_i \in \mathbb{Z}_{q_0}$ is embedded into $\mathbb{Z}_{q_1}$. Given the LWE encryption mentioned above, we first decompose $\mathbf{a}_i$ and obtain $\mathbf{a}_i = \sum_{j=0}^{\ell-1} a_{i,j} \cdot z^j$ where $a_{i,j} \in [z]$, then compute:

$$\begin{aligned} & b \cdot K^{(b)} - \sum_{i=0}^{n-1} \sum_{j=0}^{\ell-1} a_{i,j} \cdot K_{i,j}^{(a)} \\ = & \frac{b \cdot \hat{g} - \sum_{i=0}^{n-1} \sum_{j=0}^{\ell-1} a_{i,j} \cdot g_{i,j} + \Delta \cdot \delta \cdot m + \Delta \cdot e + \theta \cdot (q_1 \bmod q_0)}{f} \in R_{n_1,q_1}. \end{aligned}$$

The result is an NTRU encryption of m with secret key f and error term $g = b \cdot \hat{g} - \sum_{i=0}^{n-1} \sum_{j=0}^{\ell-1} a_{i,j} \cdot g_{i,j} + \Delta \cdot e + \theta \cdot (q_1 \bmod q_0)$ (and the scaling factor is slightly different from that in the definition in Section 2, causing extra rounding error). We use $\mathsf{SwitchToNTRU}\left((\mathbf{a}, b), K^{(b)}, \{K_{i,j}^{(a)}\}_{i \in [n], j \in [\ell]}\right)$ to denote the resulting NTRU encryption.

Transforming several LWE encryptions into one NTRU encryption. To achieve the goal of packing LWEs into one NTRU, it suffices to first transform every LWE encryption into an NTRU encryption, then perform rotation and addition. Concretely, given $r\,(\leq n)$ LWE encryptions $(\mathbf{a}^{(k)}, b^{(k)}) := (\mathbf{a}^{(k)}, \mathbf{a}^{(k)} \cdot \mathbf{s} + \delta \cdot m_k + e_k) \in \mathbb{Z}_{q_0}^{n_0} \times \mathbb{Z}_{q_0}$ $(k \in [r])$, and switching keys $K^{(b)}, K_{i,j}^{(a)}$ as above, we compute:

$$\begin{aligned} & \mathsf{PackIntoNTRU}\left(\{(\mathbf{a}^{(k)}, b^{(k)})\}_{k \in [r]}, K^{(b)}, \{K_{i,j}^{(a)}\}_{i \in [n], j \in [\ell]}\right) \\ = & \sum_{k=0}^{r-1} X^k \cdot \mathsf{SwitchToNTRU}\left((\mathbf{a}^{(k)}, b^{(k)}), K^{(b)}, \{K_{i,j}^{(a)}\}_{i \in [n], j \in [\ell]}\right). \end{aligned}$$

The result is an NTRU encryption of $\sum_{k=0}^{r-1} m_k \cdot X^k$ under secret key f.

Transforming several LWE encryptions into one VNTRU encryption. To obtain a VNTRU encryption $(\frac{g_0}{f} + B^0 \cdot m, \ldots, \frac{g_{d-1}}{f} + B^{d-1} m)^\top \in R_{n,q}^d$, it

suffices to take LWE encryptions $(\mathbf{a}^{(k)}, b^{(k)}) := (\mathbf{a}^{(k)}, \mathbf{a}^{(k)} \cdot \mathbf{s} + B^k \cdot m + e_k) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ ($k \in [d]$) and switch them into NTRU encryptions (but a slightly different form). Specifically, let $\mathbf{g}_z := (1, z, z^2, \ldots, z^{\ell-1})^\top \in \mathbb{Z}_q^\ell$ be the gadget vector with respect to modulus q and base z. Given the aforementioned LWE encryptions and switching keys

$$\tilde{K}_j^{(b)} := \frac{\hat{g}_j}{f} + z^j, \tilde{K}_{i,j}^{(a)} := \frac{g_{i,j}}{f} + z^j \cdot \mathbf{s}_i \in R_{n,q} \quad (i \in [n], j \in [\ell]),$$

we first decompose $b^{(k)}$ and $\mathbf{a}_i^{(k)}$ to obtain $b^{(k)} = \sum_{j=0}^{\ell-1} b_j^{(k)} \cdot z^j$ and $\mathbf{a}_i^{(k)} = \sum_{j=0}^{\ell-1} a_{i,j}^{(k)} \cdot z^j$. Then compute:

$$\begin{aligned}
&\mathsf{SwitchToVNTRU}\left(\{(\mathbf{a}^{(k)}, b^{(k)})\}_{k\in[d]}, \{\tilde{K}_j^{(b)}\}_{j\in[\ell]}, \{\tilde{K}_{i,j}^{(a)}\}_{i\in[n],j\in[\ell]}\right) \\
&= \left(\sum_{j=0}^{\ell-1} b_j^{(k)} \cdot \tilde{K}_j^{(b)} - \sum_{i=0}^{n-1}\sum_{j=0}^{\ell-1} a_{i,j}^{(k)} \cdot \tilde{K}_{i,j}^{(a)}\right)_{k\in[d]} \\
&= \left(\frac{\sum_{j=0}^{\ell-1} b_j^{(k)} \cdot \hat{g}_j - \sum_{i=0}^{n-1}\sum_{j=0}^{\ell-1} a_{i,j}^{(k)} \cdot g_{i,j}}{f} + B^k \cdot m + e_k\right) \in R_{n,q}^d.
\end{aligned}$$

The result is a VNTRU encryption of m with secret key f and error vector $\left(\sum_{j=0}^{\ell-1} b_j^{(k)} \cdot \hat{g}_j - \sum_{i=0}^{n-1}\sum_{j=0}^{\ell-1} a_{i,j}^{(k)} \cdot g_{i,j} + e_k \cdot f\right) \in R_{n,q}^d$.

Remark. In operation $\mathsf{SwitchToVNTRU}$, we also do gadget decomposition to $b^{(k)}$ since $\mathsf{SwitchToVNTRU}$ is *not* a time-consuming step of the protocol. Another difference from operation $\mathsf{SwitchToNTRU}$ is that now we cannot adopt smaller lattice parameters for the LWE encryptions, since the VNTRU encryptions have special scaling factors.

CMux gate with NTRU. Homomorphic computation of CMux gates is often employed to perform homomorphic selection. It was realized utilizing the RLWE-RGSW external product, for example, in VIA [LWZ26]. Now we have the NTRU-VNTRU external product, we can have CMux gates in a similar manner. Given two NTRU encryptions $c_0 := \frac{g_0 + \Delta \cdot m_0}{f} \in R_{n,q}$, $c_1 := \frac{g_1 + \Delta \cdot m_1}{f} \in R_{n,q}$ and a VNTRU encryption $C := (\frac{g_{2,0}}{f} + B^0 \cdot \nu, \ldots, \frac{g_{2,d-1}}{f} + B^{d-1} \cdot \nu)^\top \in R_{n,q}^d$ under the same secret key f, where d is the dimension of the gadget vector with respect to modulus q and decomposition base B. Let $\nu \in \{0, 1\}$ be the selection bit. Then $\mathsf{CMux}(c_0, c_1, C) = c_0 + (c_1 - c_0) \odot C$ is an NTRU encryption of $m_0 + (m_1 - m_0) \cdot \nu$ with secret key f and error term $g_0 + \nu \cdot (g_1 - g_0) + g_B^{-1}(c_1 - c_0) \cdot \mathbf{g_2}$ where $\mathbf{g_2} = (g_{2,0}, \ldots, g_{2,d-1})^\top$.

CMux can be easily extended from a 1-out-of-2 selection to a 1-out-of-2^r selection. Given 2^r NTRU encryptions $c_i^{(0)}$ ($i \in [2^r]$) with corresponding message m_i, and r VNTRU encryptions C_j ($j \in [r]$) with corresponding selection bits ν_j, do the following r rounds of CMux:

1: **for** $k = 0, 1, \cdots, r-1$, **do**

2: compute $c_i^{(k+1)} = \mathsf{CMux}\left(c_{2i}^{(k)}, c_{2i+1}^{(k)}, C_k\right)$ $(i \in [2^{r-1-k}])$.
3: **end for**

The final result c_0^r is an NTRU encryption of $m_{\sum_{j=0}^{r-1} \nu_j \cdot 2^j}$ with respect to secret key f. We abuse the notation of CMux and use $\mathsf{CMux}\left(\{c_i^{(0)}\}_{i\in[2^r]}, \{C_j\}_{j\in[r]}\right)$ to denote c_0^r, the result of the 1-out-of-2^r selection.

Transforming one NTRU encryption into one RLWE encryption. Response compression technique is applied to RLWE ciphertexts in Respire [BMW24] to reduce response size, but the form of NTRU encryption makes it hard to perform compression similar to that of RLWE. Therefore we realize transforming one NTRU encryption into one RLWE encryption, which can further be compressed. Let $\mathbf{g}_z := (1, z, z^2, \ldots, z^{\ell-1})^\top \in \mathbb{Z}_q^\ell$ be the gadget vector with respect to modulus q and base z. Suppose we have an NTRU encryption $c := \frac{g+\Delta\cdot m}{f} \in R_{n,q}$ and (RLWE-like) switching keys $\mathbf{R}_i := (a_i, a_i \cdot s + z^i \cdot f + e_i)^\top$ $(i \in [\ell])$, where $a_i \xleftarrow{R} R_{n,q}$ is a uniformly random polynomial, $s \leftarrow \chi_s$ is the RLWE secret key, $e_i \leftarrow \chi_e$ is an error polynomial. We first decompose c and obtain $c = \sum_{i=0}^{\ell-1} c_i \cdot z^i$, then compute:

$$\mathsf{SwitchToRLWE}\left(c, \{\mathbf{R}_i\}_{i\in[\ell]}\right) = \sum_{i=0}^{\ell-1} \mathbf{R}_i \cdot c_i$$

$$= \left(\sum_{i=0}^{\ell-1} a_i \cdot c_i, \sum_{i=0}^{\ell-1} a_i \cdot c_i \cdot s + g + \Delta \cdot m + \sum_{i=0}^{\ell-1} e_i \cdot c_i\right)^\top.$$

The result is an RLWE encryption of m with secret key s and error term $g + \sum_{i=0}^{\ell-1} e_i \cdot c_i$.

4 The LNRPIR Protocol

Database structure. The entries of the database are in $\mathbb{Z}_p$. We view the database $\mathbf{DB} := (\mathsf{db}_{i,j})$ as a matrix of shape $\frac{N}{2^k} \times 2^k$, where k is an integer and N is the size of the database. We aim to retrieve consecutive n_2 entries in a column from the database, which can also be viewed as an element from $\mathbb{Z}_p^{n_2}$.

Public parameters. For convenience, we introduce the notation for all parameters and distributions used in our protocol.

- Let λ be the security parameter.
- The LWE encryptions, which are to be transformed into NTRU encryptions, have dimension n_0, modulus q_0, and the same secret key and error distributions $\chi_{0,s} = \chi_{0,e} = \chi_0$. The corresponding gadget base and dimension are z_0 and ℓ_0, respectively.
- Other LWE encryptions, which are to be transformed into VNTRU encryptions, have dimension n_1, modulus q_1, and the same secret key and error

distributions $\chi'_{0,s} = \chi'_{0,e} = \chi'_0$. The corresponding gadget base and dimension are z'_0 and ℓ'_0, respectively.
- The NTRU encryptions (switching keys) have dimension n_1, modulus q_1, and the same secret key and error distributions $\chi_{1,s} = \chi_{1,e} = \chi_1$. The corresponding gadget base and dimension are z_1 and ℓ_1, respectively. The gadget base and dimension used in VNTRU encryptions are B and d, respectively.
- The RLWE encryptions, which are switching keys for transforming NTRU encryptions into RLWE encryptions, have dimension n_1, modulus q_1, and secret key distribution $\chi_{2,s}$, error distribution $\chi_{2,e}$.
- Other RLWE encryptions, which are switching keys for response compression, have dimension n_2, modulus q_2, and the same secret key and error distributions $\chi_{3,s} = \chi_{3,e} = \chi_3$. The corresponding gadget base and dimension are z_2 and ℓ_2, respectively.
- The compressed response is in $R_{n_2,q_2} \times R_{n_2,q_3}$. The moduli satisfy $q_1 \geq q_2 \geq q_3$.

Let the set of public parameters be

$$\mathsf{pp} := \{\lambda, B, d, k, \{n_i\}_{i=0}^2, p, \{q_i\}_{i=0}^3, z'_0, \ell'_0, \{z_i, \ell_i\}_{i=0}^2, \chi_0, \chi'_0, \chi_1, \chi_{2,s}, \chi_{2,e}, \chi_3\}.$$

We typically omit pp from the inputs of the algorithms in this section.

4.1 Database Preprocessing

In order to leverage response compression technique similar to that in Respire [BMW24], the database entries need to be reordered from the outset. We first present two functions used in response compression.

Definition 4.1 ([BMW24]) *The subring embedding function $\kappa : R_{n_2} \to R_{n_1}$ is the mapping*

$$\kappa\left(\sum_{i=0}^{n_2-1} f_i x^i\right) := \sum_{i=0}^{n_2-1} f_i x^{t\cdot i}.$$

κ sends the i-th coefficient of the input polynomial to the $(t \cdot i)$-th coefficient of the output polynomial, where $t = n_1/n_2$.

Definition 4.2 ([BMW24]) *The ring packing function $\Pi : R_{n_2,p}^t \to R_{n_1,p}$ takes as input a set of t polynomials $r_0, \ldots, r_{t-1} \in R_{n_2,p}$ and outputs a single element over $R_{n_1,p}$ as follows:*

$$\Pi\left(r_0, \ldots, r_{t-1}\right) := \sum_{i=0}^{t-1} x^i \cdot \kappa\left(r_i\right)$$

We regard every n_2 consecutive entries in a column of the database as the coefficients of a polynomial in $R_{n_2,p}$, and pack every $t = n_1/n_2$ consecutive such polynomials into a single polynomial in $R_{n_1,p}$ using the ring packing function. We then replace the original consecutive n_1 database entries in a column with

the coefficients of the newly constructed polynomial. What we actually do is partition each column of the database into groups of n_1 entries and perform an interleaved reordering within each group, so that the coefficients of the resulting polynomial of the final response compression step correspond exactly to n_2 consecutive entries in the original database. The detailed database setup procedure is shown in Figure 2.

$\mathsf{DB.Setup}((\mathsf{db}_{i,j}) \in (\mathbb{Z}_p)^{\frac{N}{2^k} \times 2^k})$: On input all entries of the database and the public parameters, the algorithm performs:

1. Construct a polynomial matrix $\mathbf{P} := (P_{i,j}) \in R_{n_2,p}^{\frac{N}{n_2 \cdot 2^k} \times 2^k}$, where $P_{i,j} := \sum_{r=0}^{n_2-1} db_{i \cdot n_2 + r, j} \cdot X^r$.
2. Apply the ring packing function to each group of $t = n_1/n_2$ consecutive polynomials in a column of $\mathbf{P}$ to obtain a new polynomial matrix $\mathbf{P}' := \left(P'_{i,j}\right) \in R_{n_1,p}^{\frac{N}{n_1 \cdot 2^k} \times 2^k}$, where $P'_{i,j} := \Pi\left(P_{i \cdot t, j}, P_{i \cdot t+1, j}, \ldots, P_{i \cdot t+t-1, j}\right)$
3. The preprocessed database $\mathbf{DB}' := \left(\mathsf{db}'_{i,j}\right) \in (\mathbb{Z}_p)^{\frac{N}{2^k} \times 2^k}$ is defined such that $\mathsf{db}'_{i,j}$ is the $(i \bmod n_1)$-th coefficient of $P'_{\lfloor i/n_1 \rfloor, j}$.
4. Output $\mathbf{DB}'$.

Fig. 2. Database setup algorithm

$\mathsf{Key.Gen}(\mathsf{pp})$: On input the public parameters, this algorithm generates keys used in encryption and switching keys used in homomorphic computation. In more detail:

1. Sample LWE secret keys $\tilde{\mathbf{s}}^{(1)} \leftarrow \chi_0^{n_0}$ and $\tilde{\mathbf{s}}^{(2)} \leftarrow (\chi'_0)^{n_1}$.
2. Sample RLWE secret keys $s_1 \leftarrow \chi_{2,s}$ and $s_2 \leftarrow \chi_3$.
3. Sample NTRU secret key $f \leftarrow \chi_1$. Resample f until it is invertible in R_{n_1,q_1}.
4. (a) Generate switching keys for $\mathsf{SwitchToNTRU}$.
   (b) Sample $\hat{g}, g_{i,j} \leftarrow \chi_1$ for $i \in [n_0], j \in [\ell_0]$.
   (c) Let $K^{(b)} := \frac{\hat{g}+\Delta}{f}, K^{(a)}_{i,j} := \frac{g_{i,j}+z_0^j \cdot \tilde{\mathbf{s}}_i^{(1)} \cdot \Delta}{f} \in R_{n_1,q_1}$, for $i \in [n_0]$, $j \in [\ell_0]$, where $\Delta = \lfloor q_1/q_0 \rfloor$.
5. (a) Generate switching keys for $\mathsf{SwitchToVNTRU}$.
   (b) Sample $\hat{g}_j, g_{i,j} \leftarrow \chi_1$ for $i \in [n_1], j \in [\ell'_0]$.
   (c) Let $\tilde{K}^{(b)}_j := \frac{\hat{g}_j}{f} + (z'_0)^j$, $\tilde{K}^{(a)}_{i,j} := \frac{g_{i,j}}{f} + (z'_0)^j \cdot \tilde{\mathbf{s}}_i^{(2)} \in R_{n_1,q_1}$, for $i \in [n_1], j \in [\ell'_0]$.
6. (a) Generate switching keys for $\mathsf{SwitchToRLWE}$.
   (b) Sample $a_i \xleftarrow{R} R_{n_1,q_1}$, $e_i \leftarrow \chi_{2,e}$ for $i \in [\ell_1]$.
   (c) Let $\mathbf{R}_i := (a_i, a_i \cdot s_1 + z_1^i \cdot f + e_i)^\top$, for $i \in [\ell_1]$.
7. Generate a key-switching key for response compression: $\mathbf{W} := \mathsf{Compress.Setup}(s_1, s_2) \in R_{n_1,q_2}^{2 \times \ell_2}$.
8. Output $\tilde{\mathbf{s}}^{(1)}$, $\tilde{\mathbf{s}}^{(2)}$, s_1, s_2, f, $\mathsf{SK}_1 := \{K^{(b)}, \{K^{(a)}_{i,j}\}_{i \in [n_0], j \in [\ell_0]}\}$, $\mathsf{SK}_2 := \{\{\tilde{K}^{(b)}_j\}_{j \in [\ell'_0]}, \{\tilde{K}^{(a)}_{i,j}\}_{i \in [n_1], j \in [\ell'_0]}\}$, $\mathsf{SK}_3 := \{\mathbf{R}_i\}_{i \in [\ell_1]}$, and $\mathbf{W}$.

Fig. 3. Key generation algorithm

4.2 Key Generation

The key generation procedure is described in Figure 3. For the algorithm Compress.Setup, see Section 4.5.

4.3 Query Generation

The query generation algorithm is shown in Figure 4.

Query.Gen($\mathsf{idx} \in [2^k] \times [\frac{N}{n_1 \cdot 2^k}] \times [t], \tilde{\mathbf{s}}^{(1)}, \tilde{\mathbf{s}}^{(2)}$) : On input the indices indicating the desired entries and the LWE secret keys, the algorithm performs:

1. Parse idx into (i_2, i_1, i_0), where $i_0 \in [t]$ indicates the positions of the desired n_2 coefficients in the remaining NTRU encryption after CMux, $i_1 \in [\frac{N}{n_1 \cdot 2^k}]$ represents the index of NTRU encryptions packed from LWE encryptions, $i_2 \in [2^k]$ indicates which column of $\mathbf{DB}$ is chosen.
2. Generate LWE encryptions related to i_2. We need a selection vector consisting of LWE encryptions for the first dimension. Let $m_j := \delta_{j,i_2}$ for $j \in [2^k]$ (Kronecker delta function). Generate a vector of LWE encryptions:
$$\mathbf{c} := (\mathbf{a}_j, \mathbf{a}_j \cdot \tilde{\mathbf{s}}^{(1)} + \delta \cdot m_j + e_j) \in \mathbb{Z}_{q_0}^{2^k \times (n_0+1)},$$
where $\mathbf{a}_j \xleftarrow{R} \mathbb{Z}_{q_0}^{n_0}$, $e_j \leftarrow \chi_0$ and $\delta = \lfloor q_0/p \rfloor$.
3. Generate LWE encryptions related to i_1. These encryptions are to be switched into VNTRU encryptions. Let the binary decomposition of i_1 be $i_1 = \sum_{r=0}^{\log N - \log n_1 - k} i_{1,r} \cdot 2^r$. For $r \in [\log N - \log n_1 - k]$, generate a set of d LWE encryptions:
$$\mathcal{C}_r := \{(\mathbf{a}_j, \mathbf{a}_j \cdot \tilde{\mathbf{s}}^{(2)} + B^j \cdot i_{1,r} + e_j)\}_{j \in [d]},$$
where $\mathbf{a}_j \xleftarrow{R} \mathbb{Z}_{q_1}^{n_1}$, $e_j \leftarrow \chi_0'$, B and d are the gadget base and dimension, respectively, corresponding to VNTRU encryptions.
4. Generate LWE encryptions related to i_0. These encryptions are also to be switched into VNTRU encryptions. Let $m_r := \delta_{r,i_0}$ for $r \in [t]$ (Kronecker delta function). For $r \in [t]$, generate a set of d LWE encryptions:
$$\tilde{\mathcal{C}}_r := \{(\mathbf{a}_j, \mathbf{a}_j \cdot \tilde{\mathbf{s}}^{(2)} + B^j \cdot m_r + e_j)\}_{j \in [d]},$$
where $\mathbf{a}_j \xleftarrow{R} \mathbb{Z}_{q_1}^{n_1}$, $e_j \leftarrow \chi_0'$, B and d are the gadget base and dimension, respectively, corresponding to VNTRU encryptions.
5. Output the query $\mathsf{qu} := \{\mathbf{c}, \{\mathcal{C}_r\}_{r \in [\log N - \log n_1 - k]}, \{\tilde{\mathcal{C}}_r\}_{r \in [t]}\}$.

Fig. 4. Query generation algorithm

4.4 Response Computation

In brief, upon receiving the query, the server performs the following steps to answer it. First, it multiplies the preprocessed database $\mathbf{DB}'$ by the vector of LWE encryptions $\mathbf{c}$ to obtain a new vector of LWE encryptions $\mathbf{c}'$. Then, it packs each group of consecutive n_1 terms of $\mathbf{c}'$ into an NTRU encryption leveraging

$\mathsf{PackIntoNTRU}$ and yields $\mathcal{N}$, a set of $\frac{N}{n_1 \cdot 2^k}$ NTRU encryptions. Next, it applies $\mathsf{SwitchToVNTRU}$ to $\{\mathcal{C}_r\}_{r\in[\log N - \log n_1 - k]}$ and $\{\tilde{\mathcal{C}}_r\}_{r\in[t]}$ from the query, obtaining two corresponding sets of VNTRU encryptions $\mathcal{V}_1$ and $\mathcal{V}_2$. The former is used as encryptions of the selection bits to perform CMux on $\mathcal{N}$, which will result in one final NTRU encryption $\tilde{c}$; the latter is used to recover a VNTRU encryption $\tilde{C}$ of a rotation polynomial X^{-i_0}. Finally, $\tilde{c} \odot \tilde{C}$ is the final NTRU encryption ready to be switched into an RLWE encryption and then further compressed. The response computation algorithm, omitting the compression step, is shown in Figure 5.

$\mathsf{Answer}\,(\mathbf{DB}', \mathsf{qu}, \mathsf{SK}_1, \mathsf{SK}_2, \mathsf{SK}_3)$: On input the preprocessed database, the query, and the required switching keys, this algorithm executes a sequence of subroutines to compute the response without compression:

1. Multiply the preprocessed database by the LWE encryptions. Compute $\mathbf{c}' := \mathsf{DB}' \cdot \mathbf{c}$ in $\mathbb{Z}_{q_0}$.
2. Pack the resulting LWE encryptions into NTRU encryptions. For $i \in [\frac{N}{n_1 \cdot 2^k}]$, compute:

$$\mathsf{ntru}_i := \mathsf{PackIntoNTRU}\left(\{\mathbf{c}'_{i\cdot n_1 + j}\}_{j\in[n_1]}, K^{(b)}, \{K^{(a)}_{i,j}\}_{i\in[n_0], j\in[\ell_0]}\right).$$

   Collect the results as $\mathcal{N} := \{\mathsf{ntru}_i\}_{i\in[\frac{N}{n_1\cdot 2^k}]}$.
3. Switch the remaining LWE encryptions in the query to VNTRU encryptions. For $r \in [\log N - \log n_1 - k]$, compute:

$$\mathsf{vntru}_r := \mathsf{SwitchToVNTRU}\left(\mathcal{C}_r, \{\tilde{K}^{(b)}_j\}_{j\in[\ell'_0]}, \{\tilde{K}^{(a)}_{i,j}\}_{i\in[n_1], j\in[\ell'_0]}\right).$$

   Collect the results as $\mathcal{V}_1 := \{\mathsf{vntru}_r\}_{r\in[\log N - \log n_1 - k]}$.
   For $r \in [t]$, compute:

$$\mathsf{vntru}'_r := \mathsf{SwitchToVNTRU}\left(\tilde{\mathcal{C}}_r, \{\tilde{K}^{(b)}_j\}_{j\in[\ell'_0]}, \{\tilde{K}^{(a)}_{i,j}\}_{i\in[n_1], j\in[\ell'_0]}\right).$$

   Collect the results as $\mathcal{V}_2 := \{\mathsf{vntru}'_r\}_{r\in[t]}$.
4. Perform CMux to complete selection from $\mathcal{N}$. Compute $\tilde{c} := \mathsf{CMux}\,(\mathcal{N}, \mathcal{V}_1)$, which is an NTRU encryption in R_{n_1,q_1}.
5. Recover the VNTRU encryption for rotation. Compute $\tilde{C} := \sum_{r=0}^{t-1} \mathsf{vntru}'_r \cdot X^{-r}$ where $X^{-r} = -X^{n_1 - r}$. The result is a VNTRU encryption of X^{-i_0}.
6. Rotate $\tilde{c}$ to ensure that subsequent response compression is executed correctly. Compute $\mathsf{res} := \tilde{c} \odot \tilde{C}$.
7. Switch the result into an RLWE encryption.
   Compute $\mathsf{res}' := \mathsf{SwitchToRLWE}\left(\mathsf{res}, \{\mathbf{R}_i\}_{i\in[\ell_1]}\right)$.
8. Output res'.

Fig. 5. Response computation algorithm without compression

4.5 Response Compression

We leverage the ciphertext compression technique (a combination of modulus switching and dimension reduction) similar to that in Respire [BMW24] to reduce the response size.

Dimension reduction.

Definition 4.3 ([BMW24]) *The dimension-reduction mapping* $\kappa^{-1} : R_{n_1} \to R_{n_2}$ *is*

$$\kappa^{-1}\left(\sum_{i=0}^{n_1-1} f_i x^i\right) := \sum_{i=0}^{n_2-1} f_{t\cdot i} x^i$$

This operation sends the $(t \cdot i)^{\text{th}}$ coefficient of the input ($t = n_1/n_2$) to the ith coefficient of the output, and drops all other coefficients.

Ciphertext compression. To compress an RLWE ciphertext in R_{n_1,q_1} into an RLWE ciphertext in R_{n_2,q_2}, we need a key-switching key that switches from secret key $s_1 \in R_{n_1,q_1}$ to $s_2 \in R_{n_2,q_2}$, while also applying the modulus switching technique.

Definition 4.4 ([BMW24], adapted) *Let* ℓ_2 *and* z_2 *be the dimension and the base of the gadget vector respectively. Let* $t = n_1/n_2$. *Let* $q_1 \geq q_2 \geq q_3$ *be the moduli. The key switching key* $\mathbf{W}$ *in dimension reduction is a matrix output by the following algorithm in Figure 6.*

$\mathsf{Compress.Setup}\,(s_1 \in R_{n_1,q_1}, s_2 \in R_{n_2,q_2})$: On input two associated keys (s_1, s_2),

1. Sample $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_t \xleftarrow{R} R^{\ell_2}_{n_2,q_2}$, and $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_t \leftarrow \chi_3^{\ell_2}$, where χ_3 is a given distribution over R_{n_2,q_2}.
2. Compute
$$\mathbf{a} := \Pi\left(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_t\right) \in R^{\ell_2}_{n_1,q_2}$$
and
$$\mathbf{b} := \Pi\left(s_2 \cdot \mathbf{a}_1 + \mathbf{e}_1, s_2 \cdot \mathbf{a}_2 + \mathbf{e}_2, \ldots, s_2 \cdot \mathbf{a}_t + \mathbf{e}_t\right) \in R^{\ell_2}_{n_1,q_2},$$
where Π is the ring packing function.
3. Output the key-switching key $\mathbf{W} := \left(\mathbf{a}, \mathbf{b} - (s_1 \bmod q_2) \cdot \mathbf{g}_{z_2}^{\top}\right)^{\top} \in R^{2\times\ell_2}_{n_1,q_2}$.

Fig. 6. Compression setup algorithm

Then we have the compression algorithm ([BMW24], adapted) in Figure 7.

Theorem 4.5 ([BMW24], adapted) *Let* ℓ_2 *and* z_2 *be the dimension and the base of the gadget vector, respectively. Let* $t = n_1/n_2$. *Let* $q_1 \geq q_2 \geq q_3$ *be the moduli. Let* $\mathbf{c} := (c_1, c_2)^{\top} \in R^2_{n_1,q_1}$ *be an RLWE ciphertext of* $m \in R_{n_1,p}$ *with respect to secret key* $s_1 \in R_{n_1,q_1}$ *and error* $e_{rlwe} \in R_{n_1,q_1}$. *Let* $s_2 \in R_{n_2,q_2}$ *be the target key. Let* $\mathbf{W} := \mathsf{Compress.Setup}\left(s_1, s_2\right), (\hat{c}_1, \hat{c}_2) := \mathsf{Compress}\left(\mathbf{W}, \mathbf{c}\right)$. *Let* $m' := \left\lfloor \hat{c}_2 - \left\lfloor \frac{q_3}{q_2}\left(s_2 \cdot \hat{c}_1\right)\right\rceil \right]_{q_3,p}$ *be the recovered plaintext in* $R_{n_2,p}$.

$\mathsf{Compress}(\mathbf{W}, \mathbf{c})$: On input the key-switching key $\mathbf{W} := (\mathbf{w}_1, \mathbf{w}_2)^\top \in R_{n_1,q_2}^{2\times \ell_2}$, and an RLWE ciphertext $\mathbf{c} := (c_1, c_2)^\top \in R_{n_1,q_1}^2$,

1. Compute

$$\hat{c}_1 := \kappa^{-1}\left(\mathbf{w}_1^\top \cdot \mathbf{g}_{z_2}^{-1}\left(\left\lfloor \frac{q_2}{q_1} c_1 \right\rceil \bmod q_2\right)\right) \in R_{n_2,q_2},$$

$$\hat{c}_2 := \kappa^{-1}\left(\left\lfloor \frac{q_3}{q_1} c_2 + \frac{q_3}{q_2}\mathbf{w}_2^\top \cdot \mathbf{g}_{z_2}^{-1}\left(\left\lfloor \frac{q_2}{q_1} c_1 \right\rceil \bmod q_2\right)\right\rceil \bmod q_3\right) \in R_{n_2,q_3},$$

   where κ^{-1} is the dimension-reduction mapping.
2. Output the "ciphertext" $(\hat{c}_1, \hat{c}_2) \in R_{n_2,q_2} \times R_{n_2,q_3}$.

Fig. 7. Response compression algorithm

Suppose the error distribution used in $\mathsf{Compress.Setup}$ *is subgaussian with parameter* $\sigma_{compress}$*, and the error distribution from which we sample* e_{rlwe} *is subgaussian with parameter* σ_{rlwe}*. Suppose the key distribution* $\chi_{2,s}$ *is the uniform distribution over* $(\mathbb{Z}_{q_1} \cap [-B_s, B_s])^{n_1}$*, from which the key* s_1 *is sampled. Then*

$$\hat{c}_2 - \lfloor \frac{q_3}{q_2}(s_2 \cdot \hat{c}_1)\rceil = \frac{q_3}{p}\kappa^{-1}(m) + e_1 + e_2 \in R_{n_2,q_3},$$

where

$$\|e_1\|_\infty \le \frac{1}{2}\left(2 + \frac{q_3}{q_1}(q_1 \bmod p) + \frac{q_3}{q_2} n_1 B_s\right),$$

while under the independence heuristic, e_2 *has subgaussian coefficients with variance*

$$\sigma^2 = \frac{q_3^2}{q_1^2}\sigma_{rlwe}^2 + \frac{q_3^2}{q_2^2}\ell_2 n_1 z_2^2 \sigma_{compress}^2/4.$$

When $\|e_1\|_\infty + \|e_2\|_\infty < \frac{q_3}{2p}$*, we have* $m' = \kappa^{-1}(m)$*, i.e., we recover the correct compressed plaintext.*

4.6 Construction of LNRPIR

The full description of our protocol is given in Figure 8.

Correctness. Correctness follows from the analysis in Appendix A together with the concrete values of the parameters given in Section 5.1.

Security. The security of our protocol relies on the LWE assumptions, the RLWE assumptions and the NTRU assumptions. The full proof of security is given in Appendix B.

4.7 Extension to the Batch Query Setting

As NTRU encryptions are incompatible with the dimension-reduction mapping used in response compression for RLWE encryptions, it is also difficult to realize batching for NTRU encryptions. Fortunately, we have the NTRU-to-RLWE

1. $\mathsf{DB.Setup}\left((\mathsf{db}_{i,j}) \in (\mathbb{Z}_p)^{\frac{N}{2^k} \times 2^k}\right)$: On input all entries of the database and the public parameters, the algorithm outputs a preprocessed database $\mathbf{DB}'$.
2. $\mathsf{Key.Gen}\,(\mathsf{pp})$: On input the public parameters, the algorithm outputs the LWE secret keys $\tilde{\mathbf{s}}^{(1)}$ and $\tilde{\mathbf{s}}^{(2)}$, the RLWE secret keys s_1 and s_2, the NTRU secret key f, the switching keys $\mathsf{SK}_1 = \{K^{(b)}, \{K^{(a)}_{i,j}\}_{i\in[n_0],j\in[\ell_0]}\}$, $\mathsf{SK}_2 = \{\{\tilde{K}^{(b)}_j\}_{j\in[\ell'_0]}, \{\tilde{K}^{(a)}_{i,j}\}_{i\in[n_1],j\in[\ell'_0]}\}$, $\mathsf{SK}_3 = \{\mathbf{R}_i\}_{i\in[\ell_1]}$, and the response compression key $\mathbf{W}$.
3. $\mathsf{Query.Gen}\left(\mathsf{idx} \in [2^k] \times [\frac{N}{n_1 \cdot 2^k}] \times [t], \tilde{\mathbf{s}}^{(1)}, \tilde{\mathbf{s}}^{(2)}\right)$: On input the indices indicating the desired entries and the LWE secret keys, the algorithm generates and outputs the query qu.
4. $\mathsf{Answer}\,(\mathbf{DB}', \mathsf{qu}, \mathsf{SK}_1, \mathsf{SK}_2, \mathsf{SK}_3)$: On input the preprocessed database, the query, and the required switching keys, this algorithm computes and outputs an RLWE response res' without compression.
5. $\mathsf{Compress}\,(\mathbf{W}, \mathsf{res}')$: On input the compression key and the (rotated) RLWE ciphertext, this algorithm outputs the final response $\mathsf{RES} \in R_{n_2,q_2} \times R_{n_2,q_3}$ to the query, which is a ciphertext encoded with s_2.
6. $\mathsf{Recover}(\mathsf{RES}, s_2)$: On input the compressed response and the secret key s_2, the client parses $\mathsf{RES} := (a, b)$ and computes $\mathsf{pt} := \lfloor b - a \cdot s_2 \rceil_{q_3,p}$. Let $\mathbf{pt}'$ be the coefficient vector of pt. Elements of $\mathbf{pt}' \in \mathbb{Z}_p^{n_2}$ are the retrieved entries.

Fig. 8. The LNRPIR protocol

conversion. Thus, we can leverage batching techniques designed for RLWE encryptions such as that in VIA-B [LWZ26]. The resulting trade-off between query size and computation will be inherited from LNRPIR, and the resulting response size will be similar to that in VIA-B. We do not provide further analysis here.

5 Implementation and Evaluation

In this section, we describe our parameter selection and evaluate the performance of our protocol, followed by comparisons with state-of-the-art PIR protocols.

5.1 Parameter Selection

We choose the public parameters such that the correctness error is no more than 2^{-40} (i.e., we retrieve the correct entries with probability at least $1 - 2^{-40}$). Meanwhile, the parameters ensure that each of the LWE, RLWE, and NTRU assumptions, which we rely on for security, has 128 bits of classical security. We use the lattice estimator tool [APS15] for our security estimates.

Lattice parameters. In LNRPIR, we rely on two different LWE assumptions:

– LWE with small dimension n_0 and modulus q_0, with the same secret key and error distributions $\chi_{0,s} = \chi_{0,e} = \chi_0$. We set the dimension to be $n_0 = 1024$ and q_0 to be a modulus related to the value of k, as shown in Table 1. We take χ_0 to be a discrete Gaussian distribution with parameter $\sigma_0 = 0.8$.

Table 1. Values of q_0 for different values of k.

k	9	10	11	12	13	14
q_0	39382	59839	80995	124339	124339	169614

– LWE with large dimension n_1 and modulus q_1, with the same secret key and error distributions $\chi'_{0,s} = \chi'_{0,e} = \chi'_0$. We set the dimension to be $n_1 = 2048$ and the modulus to be $q_1 = 331777 * 188417$, which is a 36-bit integer. We take χ'_0 to be a discrete Gaussian distribution with parameter $\sigma'_0 = 0.8$.

We also rely on an NTRU assumption, which is over the ring R_{n_1,q_1} and has the same secret key and error distributions $\chi_{1,s} = \chi_{1,e} = \chi_1$. The values of n_1 and q_1 have been shown above. We take χ_1 to be a discrete Gaussian distribution with parameter $\sigma_1 = 1.1$.

Moreover, we rely on two different RLWE assumptions:

– RLWE with large dimension n_1 and modulus q_1, with secret key distribution $\chi_{2,s}$ and error distribution $\chi_{2,e}$. The values of n_1 and q_1 have been shown above. We take $\chi_{2,s}$ to be the uniform distribution on the interval $[-B_s, B_s]$, where $B_s = 1$. $\chi_{2,e}$ is a discrete Gaussian distribution with parameter $\sigma_2 = 0.8$.
– RLWE with small dimension n_2 and modulus q_2, with the same secret key and error distributions $\chi_{3,s} = \chi_{3,e} = \chi_3$. We set the dimension to be $n_2 = 512$ and the modulus to be $q_2 = 2056193$, which is a 21-bit integer. We take χ_3 to be a discrete Gaussian distribution with parameter $\sigma_3 = 27.9$. The extra modulus q_3 for the compressed response is set to be 127 when $k = 9, 10, 11, 12$, and 251 when $k = 13, 14$.

Database configuration. The database is represented as a matrix in $\mathbb{Z}_p^{\frac{N}{2^k} \times 2^k}$. The modulus p is set to be 16. For each query, our protocol retrieves n_2 entries, i.e., 256 bytes of data. k is the crucial parameter controlling the trade-off between communication and computational costs, making our protocol flexible. For the experimental evaluation shown in Section 5.2, we choose k from $\{9, 10, 11, 12, 13, 14\}$, although other values can also be used to achieve different trade-offs.

Gadget parameters. We use different gadget decomposition parameters for ciphertexts under different assumptions.

– For LWE encryptions with dimension n_0 and modulus q_0, the gadget base and dimension are $z_0 = 512$ and $\ell_0 = 2$, respectively.
– For LWE encryptions with dimension n_1 and modulus q_1, the gadget base and dimension are $z'_0 = 16$ and $\ell'_0 = 9$, respectively.
– For NTRU encryptions with dimension n_1 and modulus q_1, the gadget base and dimension are $z_1 = 262144$ and $\ell_1 = 2$, respectively.
– For VNTRU encryptions with dimension n_1 and modulus q_1, the gadget base and dimension are $B = 512$ and $d = 4$, respectively.

Table 2. Comparison of LNRPIR with Pirouette [KS26], Respire [BMW24], VIA-C [LWZ26], InsPIRe [MPSY26], and SimplePIR [HHC+23] for different database sizes.

Database	Metric	Pirouette	Respire	VIA-C	LNRPIR			InsPIRe	SimplePIR
	Offline Comm.	1.2 GB	3.887MB	14.843 MB	179.479 MB	179.479 MB	179.479 MB	—	28.969 MB
$2^{18} \times 256$B (64 MB)	**Query Size**	36 B	2.297 KB	0.494 KB	1.147 KB	2.519 KB	4.234 KB	171.500 KB	28.000 KB
	Response Size	2.000 KB	2.000 KB	1.439 KB	1.748 KB	1.748 KB	1.748 KB	12.000 KB	28.000 KB
	Computation	6.500s	0.475 s	0.387 s	0.246 s	0.134 s	0.072 s	0.415 s	0.004 s
	Throughput	9.846 MB/s	134.737 MB/s	165.375 MB/s	260.163 MB/s	477.612 MB/s	888.889 MB/s	154.217 MB/s	15.625 GB/s
	Offline Comm.	1.2 GB	3.887MB	14.843 MB	179.479 MB	179.479 MB	179.479 MB	—	58.797 MB
$2^{20} \times 256$B (256 MB)	**Query Size**	36 B	4.102 KB	0.531 KB	2.194 KB	4.269 KB	8.637 KB	182.000 KB	57.000 KB
	Response Size	2.000 KB	2.000 KB	1.439 KB	1.748 KB	1.748 KB	1.748 KB	12.000 KB	58.000 KB
	Computation	8.018s	1.066 s	0.778 s	0.517 s	0.294 s	0.166 s	0.431 s	0.015 s
	Throughput	31.928 MB/s	240.150 MB/s	329.049 MB/s	495.164 MB/s	870.748 MB/s	1542.169 MB/s	593.968 MB/s	16.667 GB/s
	Offline Comm.	1.2 GB	3.887MB	14.843 MB	179.479 MB	179.479 MB	179.479 MB	—	118.672 MB
$2^{22} \times 256$B (1 GB)	**Query Size**	36 B	7.656 KB	0.578 KB	2.229 KB	4.304 KB	8.762 KB	224.000 KB	117.000 KB
	Response Size	2.000 KB	2.000 KB	1.439 KB	1.748 KB	1.748 KB	1.748 KB	12.000 KB	118.000 KB
	Computation	14.001s	2.881 s	2.179 s	2.057 s	1.151 s	0.663 s	0.521 s	0.061 s
	Throughput	73.138 MB/s	355.432 MB/s	469.940 MB/s	497.812 MB/s	889.661 MB/s	1544.495 MB/s	1965.451 MB/s	16.393 GB/s
	Offline Comm.	1.2 GB	3.887MB	14.843 MB	179.479 MB	179.479 MB	179.479 MB	—	342.605 MB
$2^{25} \times 256$B (8 GB)	**Query Size**	36 B	14.766 KB	0.663 KB	8.725 KB	17.169 KB	34.971 KB	616.000 KB	342.000 KB
	Response Size	2.000 KB	2.000 KB	1.439 KB	1.748 KB	1.809 KB	1.809 KB	12.000 KB	342.000 KB
	Computation	60.040s	17.501 s	13.258 s	5.217 s	3.385 s	2.571 s	1.364 s	0.494 s
	Throughput	136.442 MB/s	468.088 MB/s	617.891 MB/s	1540.718 MB/s	2420.089 MB/s	3186.309 MB/s	6005.865 MB/s	16.194 GB/s

– For RLWE encryptions with dimension n_2 and modulus q_2, the gadget base and dimension are $z_2 = 4$ and $\ell_2 = 11$, respectively.

PRG compression. We use a standard optimization [MW22,HHC+23,MW24, BMW24,LWZ26] to reduce the query size, wherein the client sends a PRG seed in place of the random component of the LWE encryptions in the query.

5.2 Experimental Evaluation

We implement LNRPIR using approximately 3000 lines of Rust, building upon an adapted and extended version of the spiral-rs library. The implementation is available here: https://github.com/Raiden118/LNRPIR. We use an Alibaba Cloud (Aliyun) ECS ecs.hfr8i.16xlarge instance with 64 vCPUs (Intel Xeon Platinum 6462C @ 3.3 GHz) and 512 GB of memory, and run Ubuntu 24.04 for our experiments. Our implementation uses AVX-512. The online part of all experiments is executed single-threaded to ensure a fair comparison. In all experiments, running times were averaged over 5 trials and had a standard deviation of no more than 5%.

Comparison schemes. To illustrate our new, flexible communication-computation trade-offs, we compare LNRPIR against not only the state-of-the-art single-server PIR schemes in the client-hint model: Pirouette [KS26], Respire [BMW24], and VIA-C [LWZ26], but also those in the other models: InsPIRe [MPSY26] (in the silent-preprocessing model) and SimplePIR [HHC+23] (in the database-hint model). We also present the evaluation of the recent protocol NTPIR [LH26], which also employs NTRU encryptions but uses different conversion techniques.

Macrobenchmarks. We consider several performance metrics: offline communication, online communication (query size and response size), and online server computation time. We use KB, MB, GB to denote 2^{10}, 2^{20}, 2^{30} bytes, respectively. Table 2 presents a comprehensive comparison of the performance of LNR-

Table 3. The evaluation of NTPIR [LH26].

NTPIR	$\|2^{20} \times 64\text{B}$ (64 MB)	$\|2^{22} \times 64\text{B}$ (256 MB)	$\|2^{24} \times 64\text{B}$ (1 GB)	$\|2^{27} \times 64\text{B}$ (8 GB)
Offline Comm.	—	—	—	—
Query Size	411.000KB	444.750 KB	478.500 KB	546.000 KB
Response Size	78.750 KB	78.750 KB	78.750 KB	78.750 KB
Computation	0.034s	0.101 s	0.359 s	2.378 s
Throughput	1882.353 MB/s	2534.653 MB/s	2852.368 MB/s	3444.912 MB/s

PIR under different choices of the parameter k with that of the aforementioned protocols. The throughput is defined as the ratio between the database size and online server computation time. The parameters of InsPIRe are chosen to minimize its communication cost. The experiments for LNRPIR take $k = 9, 10, 11$ for 64 MB databases, $k = 10, 11, 12$ for 256 MB databases, $k = 10, 11, 12$ for 1 GB databases, and $k = 12, 13, 14$ for 8 GB databases.

Compared with communication-efficient protocols Pirouette [KS26], Respire [BMW24], and VIA-C [LWZ26], LNRPIR achieves substantially lower computational costs while maintaining moderate communication costs. With k set to be 12, LNRPIR achieves a server throughput of 1540.718 MB/s when retrieving a 256-byte record from an 8 GB database, which is 2.541× higher than that of VIA-C and 3.355× higher than that of Respire. Meanwhile, the communication cost of LNRPIR is 10.473 KB, although this is 4.982× higher than that of VIA-C, it is still 1.601× lower than that of Respire. Furthermore, one could achieve higher server throughput by increasing k, at the cost of a larger query size.

Compared with computation-efficient protocols InsPIRe [MPSY26] and SimplePIR [HHC+23], LNRPIR achieves substantially lower communication costs while keeping computational costs low. For an 8 GB database, by setting $k = 14$, LNRPIR achieves a server throughput of 3186.309 MB/s, only 1.885× lower than that of InsPIRe and 5.204× lower than that of SimplePIR. However, the communication cost of LNRPIR is 36.780 KB, which is 17.074× lower than that of InsPIRe and 18.597× lower than that of SimplePIR. Similarly, one could achieve lower communication costs by decreasing k, at the expense of lower server throughput.

In Table 3, we present our evaluation of NTPIR [LH26]. The parameters are chosen to minimize its communication cost. We list it separately since NTPIR does not provide sufficient guidance on the choice of parameters when retrieving a 256-byte record from databases of different sizes. Thus, we only present results for retrieving 64-byte records, and the parameters for 64 MB and 256 MB databases are chosen based on our own judgment. For an 8 GB database, the communication cost of LNRPIR with $k = 14$ is 16.986× lower than that of NTPIR, while the server throughput of NTPIR is only 1.081× higher than that of LNRPIR.

Remark. The offline communication cost of LNRPIR could be effectively reduced by reducing the parameter ℓ_0' and increasing d, at the expense of a slight increase in online server computation time. (The only time-consuming step affected by parameter d is CMux.)

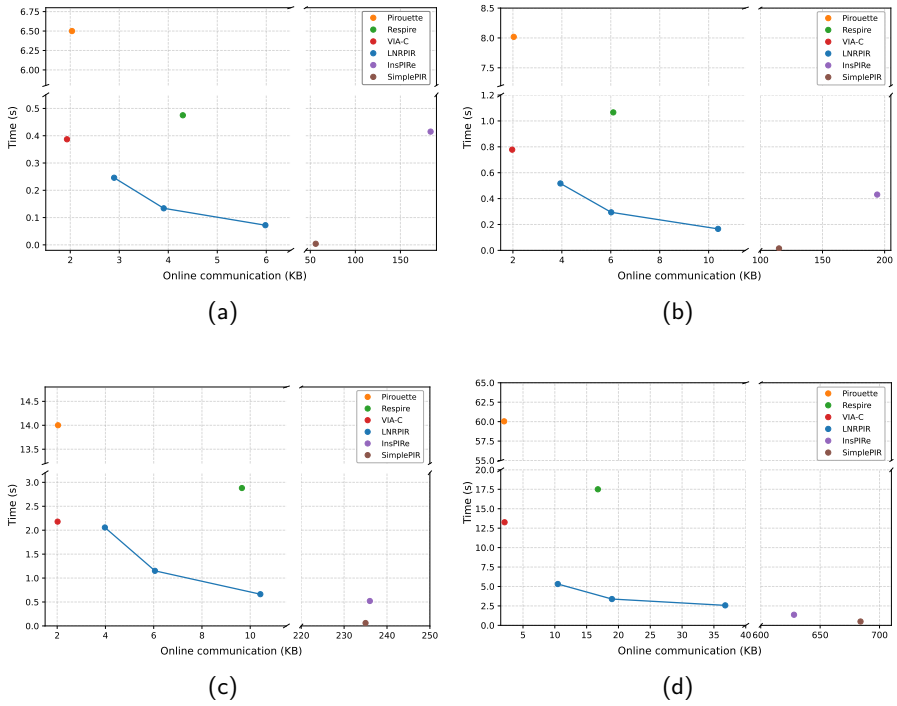

Fig. 9. Comparison of LNRPIR with Pirouette [KS26], Respire [BMW24], VIA-C [LWZ26], InsPIRe [MPSY26], and SimplePIR [HHC+23] in coordinate plots across four database sizes. Subfigures (a), (b), (c), and (d) correspond to 64 MB databases, 256 MB databases, 1 GB databases, and 8 GB databases, respectively. Here, "Time" refers to the online server computation time.

Trade-off Analysis. Figure 9 offers further insight into the trade-offs provided by LNRPIR between communication and computation. The coordinate plots also show the trade-offs of the other protocols in Table 2. One can clearly see that LNRPIR offers flexible yet efficient options, addressing a gap in the communication–computation trade-off space left by prior works.

6 Related Works

PIR was introduced by Chor et al. [CGKS95] in the multi-server setting. The single-server setting, which is the focus of our work, was introduced by Kushilevitz and Ostrovsky [KO97]. Early single-server PIR schemes were based on number-theoretic assumptions, including [CMS99] and [Cha04]. With the development of lattice-based cryptography, an increasing number of schemes have been designed leveraging lattice-based homomorphic encryption, achieving practical efficiency. The state-of-the-art lattice-based single-server schemes fall into three categories:

1. **Client-hint model.** In this case, the client sends encrypted key material to the server during the offline phase, which can then be reused for multiple queries. Schemes in this model, including SPIRAL [MW22], NTRU-PIR [XW24], Respire [BMW24], and VIA-C [LWZ26], trade higher computational costs for lower online communication costs. However, they require the server to store additional material for every client.
2. **Database-hint model.** Another class of schemes with offline communication allows the client to download hints related to the database, which are mainly used to generate queries and decrypt ciphertexts returned by the server. These schemes, including SimplePIR [HHC+23] and FrodoPIR [DPC23], incur minimal computational costs but require substantial client-side storage. Moreover, the hints must be refreshed whenever the database is updated.
3. **Silent-preprocessing model.** To overcome the aforementioned drawbacks, many recent schemes perform preprocessing without requiring offline communication. Schemes such as HintlessPIR [LMRS24], YPIR [MW24], InsPIRe [MPSY26], VIA [LWZ26], and NTPIR [LH26] implement PIR with silent preprocessing, at the cost of larger query sizes. By eliminating offline communication while maintaining relatively low computational costs, these schemes are more appealing and promising.

Another important goal in PIR research is to achieve double efficiency. Specifically, this refers to achieving both sublinear communication costs and sublinear online computational costs. [LMW23] presents a theoretical scheme attaining double efficiency, even without offline communication. Other works have also explored double-efficient PIR, but developing practical constructions remains challenging.

References

ALP+21. Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication-computation trade-offs in PIR. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 1811–1828. USENIX Association, August 11–13, 2021. 2

APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. 18

BMW24. Alexander Burton, Samir Jordan Menon, and David J. Wu. Respire: High-rate PIR for databases with small records. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 1463–1477, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press. 4, 5, 8, 11, 12, 16, 20, 21, 22, 23

CGKS95. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science*, pages 41–50, Milwaukee, Wisconsin, October 23–25, 1995. IEEE Computer Society Press. 1, 22

Cha04. Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04: 9th Australasian Conference on Information Security and Privacy*, volume 3108 of *Lecture Notes in Computer Science*, pages 50–61, Sydney, NSW, Australia, July 13–15, 2004. Springer Berlin Heidelberg, Germany. 22

CMS99. Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceedings*, Lecture Notes in Computer Science, pages 402–414. Springer, 1999. 22

DPC23. Alex Davidson, Gonçalo Pestana, and Sofía Celi. Frodopir: Simple, scalable, single-server private information retrieval. *Proc. Priv. Enhancing Technol.*, 2023(1):365–383, 2023. 23

HDCZ23. Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with tiptoe. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace, editors, *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 396–416. ACM, 2023. 1

HHC+23. Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 3889–3905, Anaheim, CA, USA, August 9–11, 2023. USENIX Association. 1, 2, 20, 21, 22, 23

KF17. Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 3–26, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland. 7

KO97. Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings 38th annual symposium on foundations of computer science*, pages 364–373. IEEE, 1997. 22

KS26. Jiayi Kang and Leonard Schild. Pirouette: Query efficient single-server PIR. *Proceedings on Privacy Enhancing Technologies*, 2026(2):215–241, 2026. 20, 21, 22

LH26. Zhihao Li and Cheng Hong. NTPIR: Efficient silent-preprocessing PIR using NTRU. In *Proceedings of the 33rd ACM Conference on Computer and Communications Security, CCS 2026*, 2026. 20, 21, 23

LMRS24. Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz. Hintless single-server private information retrieval. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part IX*, volume 14928 of *Lecture Notes in Computer Science*, pages 183–217, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. 23

LMW23. Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In Barna Saha and Rocco A. Servedio, editors, *55th Annual ACM*

Symposium on Theory of Computing, pages 595–608, Orlando, FL, USA, June 20–23, 2023. ACM Press. 23

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany. 6

LWZ26. Chenyang Liu, Xukun Wang, and Zhifang Zhang. VIA: communication-efficient single-server private information retrieval. In *IEEE Symposium on Security and Privacy, SP 2026, San Francisco, CA, USA, May 18-21, 2026*, pages 1503–1521. IEEE, 2026. 2, 3, 10, 18, 20, 21, 22, 23

MOT+11. Prateek Mittal, Femi G. Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-tor: Scalable anonymous communication using private information retrieval. In *USENIX Security 2011: 20th USENIX Security Symposium*, San Francisco, CA, USA, August 8–12, 2011. USENIX Association. 1

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany. 7

MPSY26. Rasoul Akhavan Mahdavi, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Inspire: Communication-efficient PIR with server-side preprocessing. In *IEEE Symposium on Security and Privacy, SP 2026, San Francisco, CA, USA, May 18-21, 2026*, pages 1463–1482. IEEE, 2026. 2, 3, 4, 8, 20, 21, 22, 23

MW22. Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy*, pages 930–947, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press. 8, 20, 23

MW24. Samir Jordan Menon and David J. Wu. YPIR: High-throughput single-server PIR with silent preprocessing. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association. 3, 8, 20, 23

NPVW27. Lea Nürnberger, Simon Pohmann, Mattia Veroni, and Christian Weinert. DNSPIR: Private information retrieval optimized for privacy-preserving DNS lookups. *Proceedings on Privacy Enhancing Technologies*, 2027(1), 2027. 1, 2

Pei16. Chris Peikert. A decade of lattice cryptography. *Found. Trends Theor. Comput. Sci.*, 10(4):283–424, 2016. 6

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93. ACM, 2005. 6

WXZ+25. Kaixing Wang, Binwu Xiang, Jiang Zhang, Yi Deng, and Dengguo Feng. Improved FHE bootstrapping and its applications in discretized neural networks. *IET Information Security*, 2025(1):7097351, 2025. 4, 7

XW24. Han Xia and Mingsheng Wang. Single-server PIR via NTRU-based FHE: Simpler, smaller, and faster. In *2024 IEEE European Symposium on Secu-*

rity and Privacy, pages 293–310, Vienna, Austria, July 8–12, 2024. IEEE Computer Society Press. 23

XZD+23. Binwu Xiang, Jiang Zhang, Yi Deng, Yiran Dai, and Dengguo Feng. Fast blind rotation for bootstrapping FHEs. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 3–36, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. 7

A Error analysis

In this section, we analyze the error growth in LNRPIR. Let public parameters

$$\mathsf{pp} := \{\lambda, B, d, k, \{n_i\}_{i=0}^{2}, p, \{q_i\}_{i=0}^{3}, z_0', \ell_0', \{z_i, \ell_i\}_{i=0}^{2}, \chi_0, \chi_0', \chi_1, \chi_{2,s}, \chi_{2,e}, \chi_3\}.$$

be defined as in Section 4, and let their concrete values be as specified in Section 5. We consider the variances of the error terms in each intermediate encryption, which are typically subgaussian random variables. The analysis is carried out under the independence heuristic.

1. After multiplying the preprocessed database by the LWE encryptions from the query, the resulting LWEs have error variance $\sigma_{mult}^2 := 2^k \cdot p^2 \cdot \sigma_0^2/4$.
2. Then the LWEs are packed into NTRUs, which have error variance at most $\sigma_{pack}^2 := n_1 \cdot (q_0^2 + n_0 \cdot \ell_0 \cdot z_0^2) \cdot \sigma_1^2 + (\lfloor q_1/q_0 \rfloor)^2 \cdot \sigma_{mult}^2 + 2 \cdot n_0 \cdot (q_1 \bmod q_0)^2 \cdot \sigma_0^2$.
3. Next, other LWE encryptions from the query are converted into VNTRU encryptions, which have error variance $\sigma_{vntru}^2 := (n_1 + 1) \cdot \ell_0' \cdot (z_0')^2 \cdot \sigma_1^2/4 + (\sigma_0')^2$.
4. Subsequently, we perform CMux. The resulting NTRU encryption has error variance $\sigma_{CMux}^2 := \sigma_{pack}^2 + (\log N - \log n_1 - k) \cdot d \cdot n_1 \cdot B^2 \cdot \sigma_{vntru}^2/4$.
5. Afterwards, this NTRU is rotated. The resulting NTRU encryption has error variance $\sigma_{rotated}^2 := \sigma_{CMux}^2 + t \cdot d \cdot n_1 \cdot B^2 \cdot \sigma_{vntru}^2/4$.
6. Then, it is converted into an RLWE encryption, which has error variance $\sigma_{rlwe}^2 := \sigma_{rotated}^2 + n_1 \cdot \ell_1 \cdot z_1^2 \cdot \sigma_2^2$.
7. Finally, we perform response compression and obtain the final response. By Theorem 4.5, the final error term is $e = e_1 + e_2$, where

$$\|e_1\|_\infty \leq \frac{1}{2}\left(2 + \frac{q_3}{q_1}(q_1 \bmod q_0) \cdot (q_0 \bmod p) + \frac{q_3}{q_2} n_1 B\right),$$

while e_2 has subgaussian coefficients with variance

$$\sigma^2 = \frac{q_3^2}{q_1^2}\sigma_{rlwe}^2 + \frac{q_3^2}{q_2^2}\ell_2 n_1 z_2^2 \sigma_3^2/4.$$

When all coefficients of e have absolute value smaller than $\frac{q_3}{2p}$, the decryption is correct.

B Security of LNRPIR

In this section, we show that LNRPIR satisfies query privacy under the assumptions mentioned in Section 5.1.

Theorem B.1 *Let the public parameters in LNRPIR be*

$$\mathsf{pp} := \{\lambda, B, d, k, \{n_i\}_{i=0}^{2}, p, \{q_i\}_{i=0}^{3}, z_0', \ell_0', \{z_i, \ell_i\}_{i=0}^{2}, \chi_0, \chi_0', \chi_1, \chi_{2,s}, \chi_{2,e}, \chi_3\}.$$

Let N be the size of the database. Let Q be the number of queries that the adversary makes in the query privacy game. Suppose the following assumptions hold:

1. $\mathsf{LWE}_{n_0, 2^k \cdot Q, q_0, \chi_0, \chi_0}$.
2. $\mathsf{LWE}_{n_1, (\log N - \log n_1 - k + t) \cdot d \cdot Q, q_1, \chi_0', \chi_0'}$.
3. $\mathsf{NTRU}_{n_1, 1+n_0 \cdot \ell_0, q_1, \chi_1, \chi_1}$.
4. $\mathsf{NTRU}_{n_1, (n_1+1) \cdot \ell_0', q_1, \chi_1, \chi_1}$.
5. $\mathsf{RLWE}_{n_1, \ell_1, q_1, \chi_{2,s}, \chi_{2,e}}$.
6. $\mathsf{RLWE}_{n_2, t \cdot \ell_2, q_2, \chi_3, \chi_3}$.

Then, LNRPIR satisfies query privacy for all efficient adversaries making (at most) Q queries.

Proof. We prove the theorem using a hybrid argument.

- Hybrid_0 : This is the original query privacy experiment with a bit $b \in \{0, 1\}$. To be specific, with the database and public parameters pp fixed and known to both the challenger and the adversary $\mathcal{A}$, the challenger runs $\mathsf{Key.Gen}\,(\mathsf{pp})$ to obtain LWE secret keys $\tilde{\mathbf{s}}^{(1)}$, $\tilde{\mathbf{s}}^{(2)}$, RLWE secret keys s_1, s_2, switching keys $\mathsf{SK}_1 = \{K^{(b)}, \{K_{i,j}^{(a)}\}_{i \in [n_0], j \in [\ell_0]}\}$, $\mathsf{SK}_2 = \{\{\tilde{K}_j^{(b)}\}_{j \in [\ell_0']}, \{\tilde{K}_{i,j}^{(a)}\}_{i \in [n_1], j \in [\ell_0']}\}$, $\mathsf{SK}_3 = \{\mathbf{R}_i\}_{i \in [\ell_1]}$, and a compression key $\mathbf{W}$.
  Then it sends $\mathsf{SK}_1, \mathsf{SK}_2, \mathsf{SK}_3$ and $\mathbf{W}$ to $\mathcal{A}$. When $\mathcal{A}$ queries on a pair of indices $(\mathsf{idx}_0, \mathsf{idx}_1)$, the challenger replies with $\mathsf{qu} \leftarrow \mathsf{Query.Gen}\left(\mathsf{idx}_b, \tilde{\mathbf{s}}^{(1)}, \tilde{\mathbf{s}}^{(2)}\right)$. After $\mathcal{A}$ makes up to Q queries, it outputs a bit b' as the output of the experiment.
- Hybrid_1 : This is the same experiment as Hybrid_0, except the challenger samples $K^{(b)} \xleftarrow{R} R_{n_1, q_1}$, $K_{i,j}^{(a)} \xleftarrow{R} R_{n_1, q_1}$, for $i \in [n_0], j \in [\ell_0]$.
- Hybrid_2 : This is the same experiment as Hybrid_1, except the challenger samples $\tilde{K}_j^{(b)} \xleftarrow{R} R_{n_1, q_1}$, $\tilde{K}_{i,j}^{(a)} \xleftarrow{R} R_{n_1, q_1}$, for $i \in [n_1], j \in [\ell_0']$.
- Hybrid_3 : This is the same experiment as Hybrid_2, except the challenger samples $\mathbf{R}_i \xleftarrow{R} R_{n_1, q_1}^2$, for $i \in [\ell_1]$.
- Hybrid_4 : This is the same experiment as Hybrid_3, except the challenger samples $\mathbf{W} \xleftarrow{R} R_{n_1, q_2}^{2 \times \ell_2}$.
- Hybrid_5 : This is the same experiment as Hybrid_4, except that while generating queries, the challenger samples and sends 2^k pairs $(\mathbf{a}, b) \xleftarrow{R} \mathbb{Z}_{q_0}^{n_0} \times \mathbb{Z}_{q_0}$ with $(\log N - \log n_1 - k + t) \cdot d$ pairs $(\mathbf{a}, b) \xleftarrow{R} \mathbb{Z}_{q_1}^{n_1} \times \mathbb{Z}_{q_1}$ in place of $\mathsf{qu} \leftarrow \mathsf{Query.Gen}\left(\mathsf{idx}_b, \tilde{\mathbf{s}}^{(1)}, \tilde{\mathbf{s}}^{(2)}\right)$.

Now it is easy to see that all information the adversary $\mathcal{A}$ gets is uniformly random and independent of the bit b in Hybrid_5. Thus it has no advantage in this experiment. We claim that $\forall i \in \{0,1,2,3,4\}$, the difference between the advantages of $\mathcal{A}$ in Hybrid_i and Hybrid_{i+1} is negligible. This claim finishes the proof of the theorem.

Next, we give a proof of the claim:

- Consider the only difference between Hybrid_0 and Hybrid_1.
  Under the $\mathsf{NTRU}_{n_1,1+n_0\cdot\ell_0,q_1,\chi_1,\chi_1}$ assumption, each switching key in SK_1 is computationally indistinguishable from $U \xleftarrow{R} R_{n_1,q_1}$. Thus, the difference between the advantages of $\mathcal{A}$ in Hybrid_0 and Hybrid_1 is negligible.
- Consider the only difference between Hybrid_1 and Hybrid_2.
  Under the $\mathsf{NTRU}_{n_1,(n_1+1)\cdot\ell_0',q_1,\chi_1,\chi_1}$ assumption, each switching key in SK_2 is computationally indistinguishable from $U \xleftarrow{R} R_{n_1,q_1}$. Thus, the difference between the advantages of $\mathcal{A}$ in Hybrid_1 and Hybrid_2 is negligible.
- Consider the only difference between Hybrid_2 and Hybrid_3.
  Under the $\mathsf{RLWE}_{n_1,\ell_1,q_1,\chi_{2,s},\chi_{2,e}}$ assumption, each element of SK_3 is computationally indistinguishable from $\mathbf{U} \xleftarrow{R} R_{n_1,q_1}^2$. Thus, the difference between the advantages of $\mathcal{A}$ in Hybrid_2 and Hybrid_3 is negligible.
- Consider the only difference between Hybrid_3 and Hybrid_4.
  Under the $\mathsf{RLWE}_{n_2,t\cdot\ell_2,q_2,\chi_3,\chi_3}$ assumption, we have that for $s_2 \leftarrow \chi_3$, $\mathbf{a}_i \xleftarrow{R} R_{n_2,q_2}^{\ell_2}$, and $\mathbf{e}_i \leftarrow \chi_3^{\ell_2}$, $(\mathbf{a}_i, s_2 \cdot \mathbf{a}_i + \mathbf{e}_i)^\top$ is computationally indistinguishable from $\mathbf{U} \xleftarrow{R} R_{n_2,q_2}^{2\times\ell_2}$ for all $i \in \{1,2,\ldots,t\}$. Thus $\mathbf{W}$ is computationally indistinguishable from $\mathbf{U} \xleftarrow{R} R_{n_1,q_2}^{2\times\ell_2}$, which indicates that the difference between the advantages of $\mathcal{A}$ in Hybrid_3 and Hybrid_4 is negligible.
- Consider the only difference between Hybrid_4 and Hybrid_5.
  Under the $\mathsf{LWE}_{n_0,2^k\cdot Q,q_0,\chi_0,\chi_0}$ and $\mathsf{LWE}_{n_1,(\log N-\log n_1-k+t)\cdot d\cdot Q,q_1,\chi_0',\chi_0'}$ assumptions, we have that each LWE encryption in qu is computationally indistinguishable from a uniformly random variable (over $\mathbb{Z}_{q_0}^{n_0}\times\mathbb{Z}_{q_0}$ or $\mathbb{Z}_{q_1}^{n_1}\times\mathbb{Z}_{q_1}$, respectively). Therefore, the difference between the advantages of $\mathcal{A}$ in Hybrid_4 and Hybrid_5 is negligible.