# Npir: High-Rate PIR for Databases with Moderate-Size Records

Yuliang Lin [*]
linyuliang21@nudt.edu.cn

Baosheng Wang [†]
bswang@nudt.edu.cn

Yi Wang [‡]
wangyi14@nudt.edu.cn

Rongmao Chen [§]
chromao@nudt.edu.cn

January 14, 2026

## Abstract

Private information retrieval (PIR) is a widely used technique in privacy-preserving applications that enables users to retrieve records from a database without revealing any information about their queries. This study focuses on a type of PIR that has a high ratio between the size of the record retrieved by the client and the server's response. Although significant progress has been made in high-rate PIR in recent years, the computational overhead on the server side remains rather high. This results in low server throughput, particularly for applications involving databases with moderate-size records (i.e. tens of kilobytes), such as private advertising system.

In this paper, we present Npir, a high-rate single-server PIR that is based on NTRU encoding and outperforms the state-of-the-art Spiral (Menon & Wu, S&P 2022) and NTRUPIR (Xia & Wang, EuroS&P 2024) in terms of server throughput for databases with moderate-size records. In specific, for databases ranging from 1 GB to 32 GB with 32 KB records, the server throughput of Npir is 1.50 to 2.84 times greater than that of Spiral and 1.77 to 2.55 times greater than that of NTRUPIR.

To improve server throughput without compromising the high-rate feature, we propose a novel tool called NTRU packing, which compresses the constant terms of underlying polynomials of multiple NTRU encodings into a single NTRU encoding, thereby reducing the size of the server's response. Furthermore, Npir naturally supports batch processing for moderate-size records, and can easily handle retrieving for records of varying sizes.tions, we advance secure communication protocols under challenging conditions.

**Keywords.** Private information retrieval, high rate, moderate-size records, NTRU, packing.

[*] National University of Defense Technology
[†] National University of Defense Technology
[‡] National University of Defense Technology
[§] National University of Defense Technology

# Contents

# 1 Introduction

Private information retrieval (PIR), first introduced in [19, 36], allows one or more clients to retrieve records from a database without revealing any information about the query, and has found a variety of privacy-preserving applications, including certificate transparency auditing [31, 44], web search [30], advertising [5, 27, 33], information discovery [34, 42], and more. Recently, great efforts [2,3,13–15,20,21,25,31,37,38,40–47,50,52,53,57] have been made to improve the performance of single-server PIRs via diverse techniques applied to fully homomorphic encryption.

**Moderate-size record.** Depending on the applications of PIR, the size of a database record varies from tens of bytes to hundreds of kilobytes, and is categorized as small ($\leq$10 KB), moderate or large ($\geq$100 KB) in [43]. Specifically, small records (e.g., 256 B) are selected for applications such as anonymous messaging systems [4,13] and private DNS [34,42]. In contrast, for data-intensive retrieval tasks, such as private video streaming [43], database records can be as large as 100 KB.

Notably, moderate-size records are used for privacy-preserving advertising systems [5, 27, 45] where the size of an advertisement is 20–40 KB, and for private Wikipedia [43] with a maximum article size of 30 KB. In particular, the typical network conditions considered for these two applications are mobile networks, where the client's computing power and communication bandwidth are limited. For example, the upload and download speeds for a mobile user of the private Wikipedia are 8 Mbps and 29 Mbps respectively [43]. Note that prior single-server PIRs [13, 40, 43, 45] with a high communication rate (i.e. the ratio between the size of the record retrieved by the client and the server's response) enjoy low communication overhead and computation time for the client who does not store any database-dependent hints. Thus, we concentrate on high-rate PIRs in this paper.

**Motivation.** The research into high-rate PIRs can be traced back to OnionPIR [45], the first such PIR. It was evaluated using databases with moderate-size records (i.e., 30 KB). Subsequently, Spiral [43] outperforms OnionPIR [45] in terms of rate and server throughput for all types of record size (i.e., 256 B, 30 KB and 100 KB). Note that both OnionPIR [45] and Spiral [43] have low communication overhead in the setting of moderate-size record. Following works including Respire [13] and KsPIR [40] turn to improve the communication cost and server throughput respectively in the setting of small record.

Although Spiral [43] and NTRUPIR [53] represent the current state-of-the-art in high-rate PIRs for databases with moderate-size records, they both suffer from high computational costs on the server side. In specific, generating a response for 1-32 GB databases containing $2^{15}$-$2^{20}$ 32 KB records would take 8.76-120.42 seconds and 10.39-105.13 seconds (see Table 1), respectively. This naturally raises the following question:

*Can we improve the server throughput of high-rate PIR for databases with moderate-size records?*

## 1.1 Contributions

We give an affirmative answer to above question by introducing Npir, a high-rate single-server PIR that is based on NTRU encoding and outperforms Spiral [43] in terms of server throughput for databases with moderate-size records. Figure 1 shows a simple illustration of Npir where a novel database and associated packing technique are adopted to achieve high communication rate with less computational costs. The full description of Npir is given in Figure 4.

**Database.** Unlike a regular database, which is a matrix of records, the proposed database for Npir is a matrix of polynomials from a polynomial ring $\mathcal{R}_p = \mathbb{Z}_p[X]/(X^N + 1)$. The number of rows is a multiple of $N$, the dimension of $\mathcal{R}_p$, and each record contains the coefficients of the same term from all the polynomials in a given column. This database can be considered a polynomial version of the database in KsPIR [40], enabling simultaneous first dimension processing and rotation. See Section 1.2 for more details.

**NTRU packing.** We present a novel technique called NTRU packing for extracting a record from a column of polynomials. This technique compresses $N\phi$ NTRU encodings of (rotated) polynomials into $\phi$ NTRU encodings, where $N\phi$ is the row count of the matrix. The coefficients of the underlying polynomials of the generated NTRU encodings are the constant terms of the $N\phi$ (rotated) polynomials that comprise a record. In this case, the communication rate of Npir is determined by the scaling factor of NTRU encoding[1].

**Efficient retrieval for moderate-size records.** Experimental evaluations indicate that compared to representative high-rate PIRs listed in Table 1, Npir offers more efficient retrieval for databases with moderate-size records. For databases ranging from 4 GB to 32 GB with 32 KB records, the server throughput of Npir for a single query is 2.17 to 2.84 times greater than Spiral [43], at the cost of a lower communication rate (see Figure 5). Besides, among these PIRs, Npir enjoys the least storage usage and the shortest client time. Its preprocessing time is tied with NTRUPIR [53] as the shortest (see Table 1).

**Extensions.** We present a batch version of Npir denoted by $\text{Npir}_{\mathsf{b}}$, where the server can handle a batch of queries simultaneously. When processing a batch of $T$ queries, $\text{Npir}_{\mathsf{b}}$ joins multiple queries into a single one (see Section 4.3). Additionally, we demonstrate how to use Npir to retrieve records of various sizes. When the record size changes, the server only adds the packing number that matches the new size (see Sections 1.2 and 4.3). See Section 5.4 for more performance details in these scenarios.

**Limitation: query size in communication.** Despite its advantages, Npir has certain limitations. Our solution requires more online communication: it is 2.16 times greater than Spiral, primarily due to a query size that is 5.25 times larger, whereas the response size is only 1.78 times larger. The large query size is particularly the result of NTRU-based GSW-like encryption consisting of a vector of NTRU encodings, and reducing the size of this encryption or the entire query message deserves further exploration.

## 1.2 Overview of Npir

Since Npir is based on NTRU encoding, we first take a quick glance back at related notions in [54]. Roughly, an NTRU encoding $c$ of a polynomial $u \in \mathcal{R}_p$ is $(g + \Delta \cdot u) \cdot f^{-1} \in \mathcal{R}_q$, where $f, g$ and $\Delta$ denote the invertible secret key, the error, and the scaling factor $\lfloor q/p \rceil$, respectively. The polynomial $u$ can be recovered by computing $c \cdot f/\Delta$ if the error is small enough (i.e., $\|g\|_\infty$ is less than $\Delta/2$). Besides, an NGSW encoding of the polynomial $u \in \mathcal{R}_p$ is a vector of NTRU encodings that encrypts $u$ in a GSW-like way. Given an NTRU encoding of $u \in \mathcal{R}_p$ and an NGSW encoding of $v \in \mathcal{R}_p$, performing the external product defined in [54] between these two encodings yields an NTRU encoding of $uv \in \mathcal{R}_p$.

In Npir, the database $\mathbf{D} \in \mathcal{R}_p^{N\phi \times \ell}$ is a matrix of polynomials, where $\phi$ is a positive integer and the number of rows is a multiple of $N$, and a record is indexed by a column-term pair $(\mathsf{col}, \mathsf{term})$ where $\mathsf{col} \in [\ell]$ and $\mathsf{term} \in [N]$. Also, given a fixed characteristic $p$, the size of

[1] The scaling factor is defined as $\Delta := \lfloor q/p \rceil$, where $q$ is a modulus and $p$ is a plaintext modulus. Indeed, $q$ must be aligned during communication, and the rate is not exactly equal to the scaling factor.

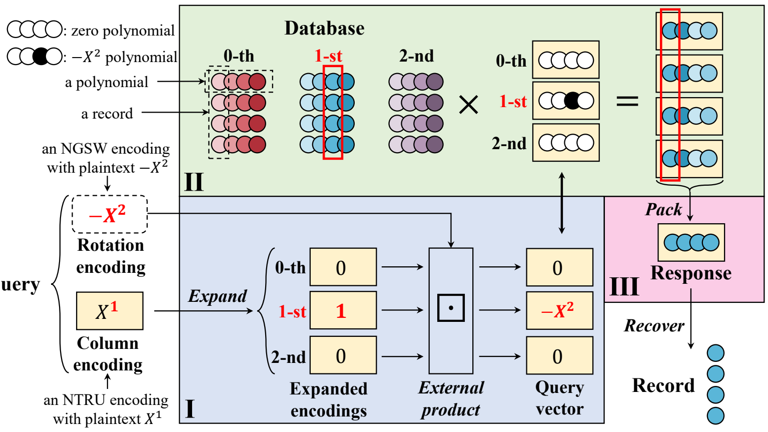

Figure 1: The overview of NPIR architecture. This example illustrates the retrieval of a record from database $\mathbf{D} \in \mathcal{R}_p^{4\times 3}$ (12 total records), at column-term index $(1, 2)$. The client's query includes a column encoding (NTRU encoding of $X^1$ for the first column) and a rotation encoding (NGSW encoding of $-X^2$, i.e., $-X^{4-2}$). The server then executes three key steps: I. Expand the column encoding into three NTRU encodings and generate the query vector via external products with the rotation encoding; II. Multiply the database by query vector; III. Pack the product result to form a response, which the client decrypts to retrieve the target record.

a record is determined by the row count $N\phi$. To retrieve a record from the database, the client sends an NTRU encoding of polynomial $X^{\mathsf{col}}$ and an NGSW encoding of polynomial $\mathsf{rot} = -X^{N-\mathsf{term}}$, denoted by column encoding and rotation encoding, respectively.

Upon receiving the query, the server first expands the column encoding. Using the coefficient expansion algorithm described in [53], the NTRU encoding of $X^{\mathsf{col}}$ can be expanded into at most $N$ NTRU encodings of the coefficients in $X^{\mathsf{col}}$ (i.e., only the coefficient of the term $X^{\mathsf{col}}$ is 1; the rest are 0). Since these expanded NTRU encodings are subsequently employed for matrix-vector multiplication with the database $\mathbf{D}$, only the first $\ell$ NTRU encodings of the coefficients in $X^{\mathsf{col}}$ are required. Then, the server computes the external product of each NTRU encoding with the rotation encoding to obtain a query vector of NTRU encodings. Note that the underlying coefficient of the $\mathsf{col}$-th NTRU encoding is the only one that is 1, while the others are 0. Consequently, the underlying polynomial of the $\mathsf{col}$-th encoding in the vector is $\mathsf{rot} = -X^{N-\mathsf{term}}$, and the others are zero polynomial. This completes the server's preparation of the query vector.

To extract a record from database $\mathbf{D}$, the server computes the matrix-vector multiplication of $\mathbf{D}$ and the query vector of NTRU encodings. The $\mathsf{col}$-th encoding is the only one that encrypts the non-zero polynomial $\mathsf{rot} = -X^{N-\mathsf{term}}$, and multiplying any polynomial by $\mathsf{rot}$ results in a new polynomial whose constant term is equal to the coefficient of the original polynomial's $X^{\mathsf{term}}$ term. The server then obtains a vector of $N\phi$ NTRU encodings of polynomials, the constant terms of which constitute the targeted record. Clearly, it would be impractical and unnecessary to return all $N\phi$ NTRU encodings to the client. To address this issue, the server performs NTRU packing on these $N\phi$ NTRU encodings to obtain a set of $\phi$ NTRU encodings. Then, the client can retrieve the record from this set.

Our NTRU packing mechanism adapts the LWEs-to-RLWE ring packing paradigm from [18] to the NTRU setting. Specifically, the LWEs-to-RLWE ring packing in [18] relies on the field trace property $\mathrm{Tr}_{\mathcal{R}_q/\mathbb{Z}_q}$, using an FFT-style algorithm. Given a polynomial $u \in \mathcal{R}_q$, $\mathrm{Tr}_{\mathcal{R}_q/\mathbb{Z}_q}$ sums all automorphisms of $u$ in a specific Galois group (i.e., $\mathrm{Tr}_{\mathcal{R}_q/\mathbb{Z}_q}(u) = \sum_\kappa \tau_\kappa(u)$ where $\tau_\kappa : u(X) \to u(X^\kappa)$). A key property is that $\mathrm{Tr}_{\mathcal{R}_q/\mathbb{Z}_q}(u)$ preserves the constant term (scaled by

$N$) while vanishing all non-constant terms. We apply this property to the automorphism for NTRU encoding [54], and present NTRU packing, which processes $N$ NTRU encodings in an FFT-style structure [18].The resulting encoding retains only the constant terms of the inputs. NTRU packing aligns with our database structure by design: it aggregates $N\phi$ NTRU encodings (whose constant terms form a target record) into $\phi$ encodings, resulting in plaintext containing solely the desired record.

## 2 Background and Definitions

**Notation.** Let $a, b, n$ be positive integers. We define the integer interval $[a : b] = \{a, a+1, \ldots, b\}$ for $a < b$ and $[n] = \{0, 1, \cdots, n-1\}$. For a polynomial $x$, let $x_i$ denote its $i$-th coefficient, and let $x \cdot y$ denote polynomial multiplication. For real numbers, $\lfloor x \rceil$ denotes rounding, $\|x\|_\infty$ is the infinite norm, and $|x|$ is the absolute value. These operations extend to polynomials in a coefficient-wise fashion. We use $\leftarrow\!\!\$$ to denote sampling from a distribution or a set.

### 2.1 Private Information Retrieval

As illustrated in Figure 2, we adopt the two-message single-server private information retrieval (PIR) definition in the *client-hint* model [19], where the server stores a reusable client query key offline. Following [13], we allow for server-side database preprocessing. This model excels in low communication overhead and client-side computational efficiency, making it well-suited for resource-constrained clients (e.g., limited bandwidth and computing power).

**Definition 1** (Private Information Retrieval, adapted from [13, 43])**.** *Let $\lambda$ be a security parameter and $\mathcal{D}$ be a database with $n$ records. A two-message single-server private information retrieval (PIR) in the client-hint model consists of the following five algorithms:*

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{qk})$*: The setup algorithm takes as input a security parameter $\lambda$. It returns a public parameter* $\mathsf{pp}$ *and a query key* $\mathsf{qk}$*.*
- $\mathsf{SetupDB}(1^\lambda, \mathcal{D}) \to \mathbf{D}$*: The database setup algorithm takes as input a security parameter $\lambda$ and a database $\mathcal{D}$ with $n$ records. It returns a pre-processed database* $\mathbf{D}$*.*
- $\mathsf{Query}(\mathsf{qk}, \mathsf{ind}) \to \mathsf{qu}$*: The query generation algorithm takes as input the query key* $\mathsf{qk}$ *and an index* $\mathsf{ind}$*. It returns a query* $\mathsf{qu}$*.*
- $\mathsf{Response}(\mathsf{pp}, \mathbf{D}, \mathsf{qu}) \to \mathsf{resp}$*: The response generation algorithm takes as input the public parameter* $\mathsf{pp}$*, a pre-processed database* $\mathbf{D}$*, and the query* $\mathsf{qu}$*. It returns a response* $\mathsf{resp}$*.*
- $\mathsf{Recover}(\mathsf{qk}, \mathsf{resp}) \to d$*: The recovery algorithm takes as input the query key* $\mathsf{qk}$ *and the response* $\mathsf{resp}$*. It returns a record $d$.*

*Furthermore, this PIR fulfills the following properties:*

- ***Correctness****: Take as input a security parameter $\lambda$, a database $\mathcal{D}$ with $n$ records, and an index* $\mathsf{ind} \in [n]$*. We say that the PIR scheme is correct if there exists a negligible function* $\mathsf{negl}(\lambda)$*, such that it satisfies that:*

$$\Pr\left[ d \neq \mathcal{D}_{\mathsf{ind}} : \begin{array}{c} (\mathsf{pp}, \mathsf{qk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathbf{D} \leftarrow \mathsf{SetupDB}(1^\lambda, \mathcal{D}) \\ \mathsf{qu} \leftarrow \mathsf{Query}(\mathsf{qk}, \mathsf{ind}) \\ \mathsf{resp} \leftarrow \mathsf{Response}(\mathsf{pp}, \mathbf{D}, \mathsf{qu}) \\ d \leftarrow \mathsf{Recover}(\mathsf{qk}, \mathsf{resp}) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

*If there exists an error $\delta$ such that* $\Pr\left[d \neq \mathcal{D}_{\mathsf{ind}}\right] \leq \delta$*, we say that the PIR scheme is* $\delta-$*correct.*

- ***Query privacy:*** *Take as input a security parameter $\lambda$ and a database $\mathcal{D}$ with $n$ records. We say that the PIR scheme has query privacy, if for any* PPT *adversary $\mathcal{A}$, $b \leftarrow\!\!\$ \{0,1\}$ and* $(\mathsf{pp}, \mathsf{qk}) \leftarrow \mathsf{Setup}(1^\lambda)$*, there exists a negligible function* $\mathsf{negl}(\lambda)$*, such that satisfies that:*
$$\left|\Pr\left[\mathcal{A}^{\mathcal{O}_b(\mathsf{qk},\cdot,\cdot)}(1^\lambda, \mathsf{pp}) = b\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda),$$
*where the oracle* $\mathcal{O}_b(\mathsf{qk},\cdot,\cdot)$ *takes as input two different indexes* $(\mathsf{ind}_0, \mathsf{ind}_1)$*, and returns a query* $\mathsf{qu} \leftarrow \mathsf{Query}(\mathsf{qk}, \mathsf{ind}_b)$*.*

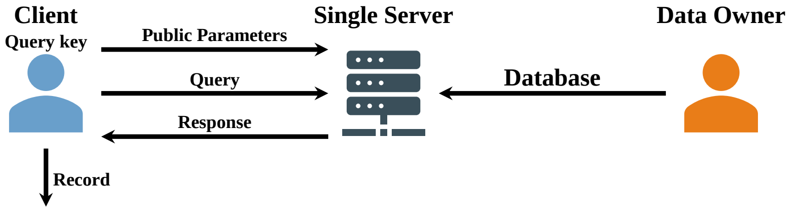

Figure 2: The framework of two-message single-server PIR in client-hint model.

## 2.2 Discrete Gaussians

We review the discrete Gaussian distribution and its variant, the *discrete sub-Gaussian distribution*, for our proposed schemes. More details can be found in [49].

We employ the discrete Gaussian distribution $D_{\mathbb{Z},\sigma}$ over $\mathbb{Z}$ with mean 0 and parameter $\sigma$, where the probability mass function is defined below:

$$\Pr[X = x] := \frac{\rho_\sigma(x)}{\sum_{y\in\mathbb{Z}} \rho_\sigma(y)},$$

where $\rho_\sigma(x) = \exp\left(-\pi x^2/\sigma^2\right)$. Moreover, the sub-Gaussian distribution satisfies $\forall y \geq 0, \Pr[|X| > y] \leq 2\exp\left(-\pi y^2/\sigma^2\right)$ and offers key properties:

- **Scaling**: For sub-Gaussian $X$ with parameter $\sigma$ and $r \in \mathbb{R}$, $rX$ is sub-Gaussian with parameter $|r|\sigma$.

- **Summation**: The sum of $d$ independent sub-Gaussian $X_i$ with parameters $\sigma_i$ yields a sub-Gaussian distribution with parameter $\sqrt{\sum_{i=0}^{d-1}\sigma_i^2}$.

## 2.3 Polynomial Rings

We use the cyclotomic polynomial ring, denoted by $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$, where $N$ is a power of two. For a positive integer $q$, we define $\mathcal{R}_q := \mathcal{R}/q\mathcal{R} = \mathbb{Z}_q[X]/(X^N + 1)$. We provide two functions on $\mathcal{R}$ as follows:

- **The coefficient function** $\mathsf{Coeff}(\cdot)$: taking as input a polynomial $f$ from $\mathcal{R}$, $\mathsf{Coeff}(f)$ returns a row vector $[f_0, f_1, \cdots, f_{N-1}]$.

- **The anticirculant function** $\mathcal{M}(\cdot)$: taking as input a polynomial $f$ from $\mathcal{R}$, $\mathcal{M}(f)$ returns a matrix

$$\mathcal{M}(f) := \begin{bmatrix} f_0 & f_1 & \cdots & f_{N-1} \\ -f_{N-1} & f_0 & \cdots & f_{N-2} \\ -f_{N-2} & -f_1 & \cdots & f_{N-3} \\ \vdots & \vdots & \ddots & \vdots \\ -f_1 & -f_2 & \cdots & f_0 \end{bmatrix},$$

which is the anti-circulant matrix representation. Moreover, the above two functions satisfy that for all $f, g \in \mathcal{R}$, we have $\mathsf{Coeff}(f \cdot g) = \mathsf{Coeff}(f) \cdot \mathcal{M}(g)$.

We introduce the conclusion provided in [43], which combines sub-Gaussian distributions and the infinite norm, and is commonly used to analyze the error generated by modulus switching in Section 2.5.

**Lemma 1** (A bound for sub-Gaussian and infinite norm, [43])**.** *Let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. Assume that $g \in \mathcal{R}$ satisfies $\|g\|_\infty \leq \beta$ and $f \in \mathcal{R}$ is sampled from a sub-Gaussian distribution with parameter $\sigma$. For the polynomial $h = f \cdot g$, we find that the distribution of each coefficient in $h$ is a sub-Gaussian distribution with parameter $\sqrt{N}\beta\sigma$.*

Next, we define the *gadget* for the polynomial ring. Given a modulus $q$ and a decomposition basis $B < q$, we define the gadget vector as $\mathbf{g}_B = [1, B, \cdots, B^{t-1}]$, where $t = \lceil \log_B q \rceil$. For a polynomial ring $\mathcal{R}_q$, we define an inverse function $\mathbf{g}_B^{-1} : \mathcal{R}_q \to \mathcal{R}_B^t$ that decomposes each coefficient and returns $t$ ordered polynomials. For a polynomial $u \in \mathcal{R}_q$, we have $\mathbf{g}_B \cdot \mathbf{g}_B^{-1}(u) = u$.

## 2.4 NTRU and Its Applications

Here, we review the definition of NTRU and its associated concepts. First, we define the decisional NTRU problem, which requires the parameter set $(N, q, \chi)$ and serves as a basis assumption.

**Definition 2** (Decisional NTRU problem, [35, 54])**.** *Given integers $N, q > 0$, denote that $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ are two polynomial rings. Let $\chi$ be a sub-Gaussian distribution over $\mathcal{R}$ with parameter $\sigma_\chi$. The decisional NTRU problem is to distinguish between $g \cdot f^{-1} \in \mathcal{R}_q$ and a uniform sample from $\mathcal{U}(\mathcal{R}_q)$, where $f$ and $g$ are sampled from $\chi$, and $f$ is invertible.*

**NTRU encoding.** This paper uses an NTRU-based Regev-like encoding [54], denoted by NTRU encoding. Notably, for an invertible secret key $f \in \chi$ and a plaintext $u \in \mathcal{R}_p$ where $p$ is the plaintext modulus, NTRU encoding is defined as:

$$\mathsf{NTRU}_f(u) := (g + \Delta \cdot u) \cdot f^{-1} \in \mathcal{R}_q,$$

where $g$ is sampled from $\chi$ and $\Delta$ is the scaling factor $\lfloor q/p \rfloor$.

Additionally, NGSW encoding [54] is an NTRU-based variant of GSW encoding. Specifically, for a decomposition base $B_g \in \mathbb{N}$ with $t_g = \lceil log_{B_g} q \rceil$ and a plaintext $v \in \mathcal{R}_q$, the NGSW encoding can be represented as:

$$\mathsf{NGSW}_f(B_g, v) := (\frac{g_0}{f} + B_g^0 \cdot v, \cdots, \frac{g_{t_g-1}}{f} + B_g^{t_g-1} \cdot v) \in \mathcal{R}_q^{t_g}.$$

Given an NTRU encoding and an NGSW encoding, we can perform an external product in [54] defined as:

$$\begin{aligned} \mathsf{NGSW}_f(B_g, v) \boxdot_{B_g} \mathsf{NTRU}_f(u) := &\ \mathsf{NGSW}_f(B_g, v) \\ &\cdot \mathbf{g}_{B_g}^{-1}(\mathsf{NTRU}_f(u)) = \mathsf{NTRU}_f(u \cdot v). \end{aligned}$$

**Algorithm 1:** NTRU Coefficient Expansion

```
Input: the expansion key ck, a decomposition base $B_{ce}$, an NTRU encoding $c$, and the
       expansion length $\ell$
Output: a vector of NTRU encodings $\{\hat{c}_j\}_{j\in[0,2^r-1]}$
set $\hat{c}_0 \leftarrow c$ and $r \leftarrow \lceil \log_2 \ell \rceil$
split ck into $\{\mathbf{W}_\kappa\}_{(\kappa=N/2^j+1)\wedge(j\in[0,r-1])}$
for $i = 0$ to $r-1$ do
    $\kappa \leftarrow N/2^i + 1$
    for $j = 0$ to $2^i - 1$ do
        $\tilde{c_j} \leftarrow c_j \cdot X^{-2^i}$
        $\hat{c}_j \leftarrow c_j + \mathsf{Automp}(\mathbf{W}_\kappa, c_j, \tau_\kappa, B_{ce})$
        $\hat{c}_{j+2^i} \leftarrow \tilde{c_j} + \mathsf{Automp}(\mathbf{W}_\kappa, \tilde{c_j}, \tau_\kappa, B_{ce})$
return $\{\hat{c}_j\}_{j\in[0,2^r-1]}$
```

Assume that NTRU encoding and NGSW encoding have error distributions with parameters $\sigma_\chi$ and $\sigma_{gsw}$, respectively. After external product, the error distribution has a parameter of $\sqrt{\frac{1}{12}tNB^2\sigma_{gsw}^2 + \|v\|_2^2\sigma_\chi^2}$, as analyzed in [54].

**Automorphisms for NTRU encoding.** Given a polynomial $u = u(X) \in \mathcal{R}_q$, we define a function $\tau_\kappa : \mathcal{R}_q \to \mathcal{R}_q$ to compute $u(X) \to u(X^\kappa)$ in [18], where $\kappa$ belongs to the invertible residues modulo $2N$. We simplify the property described in Section 1.2 for polynomial ring as:

$$\textstyle\sum_{i\in[N]} \tau_{2i+1}(X^j) = \begin{cases} N & j = 0 \\ 0 & 0 < j < N \end{cases}. \tag{1}$$

Based on this, [54] proposes an automorphism for NTRU encoding. Given a security parameter $\lambda$, an error distribution $\chi$, an invertible secret key $f \in \chi$, a function $\tau_\kappa$, and a decomposition base $B \in \mathbb{N}$, the automorphism for NTRU encoding consists of two algorithms:

- $\mathsf{AutoSetup}(1^\lambda, f, \tau_\kappa, B)$: The setup algorithm takes as input a security parameter $\lambda$, a secret key $f$, a function $\tau_\kappa$, and a decomposition base $B$. It returns an automorphism key, which is given by: $\mathbf{W}_\kappa = ((g_1, \cdots, g_t) + \tau_\kappa(f) \cdot \mathbf{g}_B) \cdot f^{-1}$, where $t = \lceil \log_B q \rceil$ and $g_1, \cdots, g_t \xleftarrow{\$} \chi$.
- $\mathsf{Automp}(\mathbf{W}_\kappa, c, \tau_\kappa, B)$: The automorphism algorithm takes as input the automorphism key $\mathbf{W}_\kappa$, an NTRU encoding $c \in \mathcal{R}_q$, a function $\tau_\kappa$, and a decomposition base $B \in \mathbb{N}$. It returns an NTRU encoding, denoted by $\tilde{c} \leftarrow \mathbf{W}_\kappa \boxdot_{B_g} \tau_\kappa(c)$.

Assume that NTRU encoding and automorphism key have error distributions with parameters $\sigma_\chi$ and $\sigma_{au}$, respectively. After automorphism, the error distribution has a parameter of $\sqrt{\frac{1}{12}tNB^2\sigma_{au}^2 + \sigma_\chi^2}$, as analyzed in [54].

**NTRU coefficient expansion.** The coefficient expansion algorithm [3, 17, 43, 53] enables a client to send each query with only one or a few encodings. Notably, [53] extends this technique to NTRU encoding. Specifically, the coefficient expansion on NTRU encoding consists of two algorithms:

- $\mathsf{CESetup}(1^\lambda, f, B_{ce}, \ell)$: The setup algorithm takes as input a security parameter $\lambda$, a secret key $f$, a decomposition base $B_{ce}$, and an expansion length $\ell$ with $r \leftarrow \lceil \log_2 \ell \rceil$. It returns an expansion key $\mathsf{ck} = (\mathbf{W}_{N+1}, \mathbf{W}_{N/2+1}, \cdots, \mathbf{W}_{N/2^{r-1}+1})$, where $\mathbf{W}_\kappa \leftarrow \mathsf{AutoSetup}(1^\lambda, f, \tau_\kappa, B_{ce})$.

- $\mathsf{CoeffExpand}(\mathsf{ck}, B_{ce}, c, \ell)$: The expansion algorithm takes as input the expansion key $\mathsf{ck}$, a decomposition base $B_{ce}$, an NTRU encoding $c$ with plaintext $m_0 + \cdots + m_{\ell-1} \cdot X^{\ell-1}$ and an expansion length $\ell$ with $r \leftarrow \lceil \log_2 \ell \rceil$. It returns a vector of NTRU encodings $\{\hat{c}_j\}_{j\in[0,2^r-1]}$ via Algorithm 1, where $\hat{c}_j$ is an NTRU encoding for $2^r \cdot m_j$.

Assume that NTRU encoding and expansion key have error distributions with parameters $\sigma_\chi$ and $\sigma_{ce}$, respectively. After coefficient expansion, the error distribution has a parameter of $\sqrt{\frac{1}{3}(2^r)^2 t_{ce} N B_{ce}^2 \sigma_{ce}^2 + (2^r)^2 \sigma_\chi^2}$, as analyzed in [43, 53].

## 2.5 Modulus Switching

On Definition 2, we review the modulus switching technique in [53]. Given a modulus $q$, an NTRU encoding $c \in \mathcal{R}_q$, and a scaling modulus $q_1$, the technique operates as follows:

$$\bar{c} \leftarrow \mathsf{Modswitch}(c, q, q_1) := \lfloor \frac{q_1}{q} c \rceil.$$

Assume that NTRU encoding has an error distribution with parameter $\sigma_\chi$. After modulus switching, $\bar{c}$ is an NTRU encoding with error $e_1 + e_2$, where $e_1$ satisfies that its infinity norm is at most $\|e_1\|_\infty \leq \frac{1}{2}$ and $e_2$ has a parameter of $(\frac{q_1}{q})\sigma_\chi$.

## 2.6 LWEs-to-RLWE Ring Packing

We revisit the LWEs-to-RLWE ring packing technique presented in [18], which compresses multiple LWE ciphertexts into a single RLWE ciphertext. Furthermore, [44] introduces a security property for the technique, pseudorandomness given the public key, based on a standard circular security variant. The formal definition is provided below.

**Definition 3** (LWEs-to-RLWE ring packing, [18])**.** *Let $\lambda$ be a security parameter and $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ be a polynomial ring where $N$ is a power of two. Let $q$ be the modulus and $\chi$ be the error distribution. LWEs-to-RLWE ring packing consists of the following two algorithms:*

- $\mathsf{Setup}(1^\lambda, s, B_{pk}) \to \mathsf{pk}$*: The setup algorithm takes as input a security parameter $\lambda$, a secret key $s$, and a decomposition base $B_{pk}$. It returns a packing key* $\mathsf{pk}$*.*
- $\mathsf{Packing}(\mathsf{pk}, \mathbf{C}) \to \hat{c}$*: The packing algorithm takes as input the packing key* $\mathsf{pk}$ *and a matrix $\mathbf{C}$ of multiple LWE ciphertexts. It returns an RLWE ciphertext $\hat{c}$.*

*Furthermore, the ring packing technique needs to fulfill the following properties:*

- ***Correctness****: Take as input a security parameter $\lambda$, a secret key $s$, a matrix $\mathbf{C} = [\mathbf{c}_0 \mid \mathbf{c}_1 \mid \cdots \mathbf{c}_{N-1}] \in \mathbb{Z}_q^{(N+1)\times N}$, and a decomposition base $B_{pk}$. Define the secret vector as* $\mathsf{Coeff}(s) = (s_0, s_1, \cdots, s_{N-1})$ *and* $[-\mathsf{Coeff}(s) \mid 1] \cdot \mathbf{c}_i = v_i \in \mathbb{Z}_q$*. After calling* $\mathsf{pk} \leftarrow \mathsf{Setup}(1^\lambda, s, B_{pk})$ *and* $\hat{c} \leftarrow \mathsf{Packing}(\mathsf{pk}, \mathbf{C})$*, we have that $\hat{c} \cdot s \approx N \sum_{i\in[N]} v_i X^i$, where we ignore the error.*
- ***Pseudorandomness given the public key****: Taking as input a security parameter $\lambda$, we say that the LWEs-to-RLWE ring packing is pseudorandom given the public key, if for any* $\mathsf{PPT}$ *adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *satisfying that*

$$\left| \Pr\left[ \begin{array}{c} \mathcal{A}(1^\lambda, \mathsf{pk}, \mathbf{a}, \mathbf{t}_b) \\ = b \end{array} : \begin{array}{c} b \leftarrow\!\!\$ \{0,1\}, s \leftarrow\!\!\$ \chi \\ \mathbf{a} \leftarrow\!\!\$ \mathcal{R}_q^m, \mathbf{e} \leftarrow\!\!\$ \chi^m \\ \mathsf{pk} \leftarrow \mathsf{Setup}(1^\lambda, s, B) \\ \mathbf{t}_0 \leftarrow s \cdot \mathbf{a} + \mathbf{e} \\ \mathbf{t}_1 \leftarrow\!\!\$ \mathcal{R}_q^m \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

# 3 Packing for NTRU Encodings

In this section, we present the first NTRU-oriented packing scheme that efficiently compresses multiple NTRU encodings. The complete scheme is provided in Figure 3.

## 3.1 Syntax

Our work extends the packing model from LWEs-to-RLWE ring packing [18] to NTRU encoding. While [18] employs a matrix-to-polynomial transformation, applying the matrix form of NTRU directly results in incompatibility with the ring-packing model due to the invertible secret matrix $\mathbf{F}$. To address this incompatibility, we introduce a special anticirculant matrix $\mathbf{F} = \mathcal{M}(f)$, derived from a polynomial $f$. This adaptation allows us to use NTRU encoding to design the NTRU packing technique. Specifically, NTRU packing takes multiple NTRU encodings as input and outputs a single encoding.

Below, we provide a formal definition of NTRU packing.

**Definition 4** (NTRU packing)**.** *Let $\lambda$ be a security parameter and $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ be a polynomial ring where $N$ is a power of two. Let $q$ be the modulus and $\chi$ be the error distribution. The NTRU packing consists of the following two algorithms:*

- $\mathsf{NPSetup}(1^\lambda, f, B_{pk}) \to \mathsf{pk}$*: The setup algorithm takes as input a security parameter $\lambda$, a secret key $f$, and a decomposition base $B_{pk}$. It returns a packing key* $\mathsf{pk}$*.*
- $\mathsf{NPacking}(\mathsf{pk}, \mathbf{C}) \to \hat{c}$*: The packing algorithm takes as input the packing key* $\mathsf{pk}$ *and a vector $\mathbf{C}$ of $N$ NTRU encodings. It returns a single NTRU encoding $\hat{c}$.*

*Furthermore, the NTRU packing needs to fulfill the following properties:*

- ***Correctness****: Take as input a security parameter $\lambda$, a secret key $f$, a vector $\mathbf{C} = [c_0 \mid c_1 \mid \cdots c_{N-1}] \in \mathcal{R}_q^N$ with $c_i \cdot f = v_{i,0} + \cdots + v_{i,N-1} \cdot X^{N-1}$ and a decomposition base $B_{pk} \in \mathbb{N}$. After calling $\mathsf{pk} \leftarrow \mathsf{NPSetup}(1^\lambda, f, B_{pk})$ and $\hat{c} \leftarrow \mathsf{NPacking}(\mathsf{pk}, \mathbf{C})$, we have that*

$$\hat{c} \cdot f \approx N \cdot \sum_{i \in [N]} v_{i,0} \cdot X^i,$$

*where we ignore the error.*

- ***Pseudorandomness given the public key****: Taking as input a security parameter $\lambda$ and a sample size $m$, we say that the NTRU packing is pseudorandom given the public key if for any* $\mathsf{PPT}$ *adversary $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *satisfying such that*

$$\left| \Pr\left[ \begin{array}{c} \mathcal{A}(1^\lambda, \mathsf{pk}, \mathbf{t}_b) \\ = b \end{array} : \begin{array}{c} b \leftarrow \{0,1\}, \mathbf{g} \leftarrow\!\$\, \chi^m \\ f \leftarrow\!\$\, \chi \textit{ with } f^{-1} \\ \mathsf{pk} \leftarrow \mathsf{NPSetup}(1^\lambda, f, B_{pk}) \\ \mathbf{t}_0 \leftarrow \mathbf{g} \cdot f^{-1}, \mathbf{t}_1 \leftarrow\!\$\, \mathcal{R}_q^m \end{array} \right] - \frac{1}{2} \right| \le \mathsf{negl}(\lambda).$$

## 3.2 Construction

Here, we provide a complete description of our NTRU packing technique. This technique uses the automorphism for NTRU encoding described in Section 2.4. Inspired by the structure of LWEs-to-RLWE ring packing, we develop an FFT-style packing structure. Specifically, NTRU

```
The NTRU packing

NPSetup($1^\lambda, f, B_{pk}$) The algorithm does:
    1. Key generation: For $i \in [\kappa]$,
        (a) sample $g_i \leftarrow\$ \chi$ for $i \in \left[t_{pk} = \left\lceil \log_{B_{pk}} q \right\rceil\right]$;
        (b) execute $\mathbf{W}_i \leftarrow \mathsf{AutoSetup}(1^\lambda, f, \tau_{2^i+1}, B_{pk})$.
    2. Return $\mathsf{pk} \leftarrow \{\mathbf{W}_i\}$.

NPacking($\mathsf{pk}, \mathbf{C}$) The algorithm does:
    Return $\mathsf{NPHelp}(\mathsf{pk}, \mathbf{C} = [c_0, c_1, \cdots, c_{N-1}])$.

NPHelp($\mathsf{pk}, [c_0, \cdots, c_{2^\eta-1}]$) The sub-algorithm does:
    1. If $\eta = 1$, return $c_0$ directly.
    2. Recursive calling: Compute
        $c_{even} \leftarrow \mathsf{NPHelp}(\mathsf{pk}, [c_0, c_2, \cdots, c_{2^\eta-2}])$,
        $c_{odd} \leftarrow \mathsf{NPHelp}(\mathsf{pk}, [c_1, c_3, \cdots, c_{2^\eta-1}])$.
    3. Rotation processing: Calculate
        $\alpha \leftarrow c_{even} + X^{N/2^\eta} \cdot c_{odd}$,
        $\beta \leftarrow c_{even} - X^{N/2^\eta} \cdot c_{odd}$.
    4. $\tau$ layer: Return $\alpha + \mathsf{Automp}(\mathbf{W}_\eta, \beta, \tau_{2^\eta+1}, B_{pk})$.
```

Figure 3: The construction of the NTRU packing.

packing first executes recursive calls and then processes the rotation and $\tau$ layer. To simplify the process, we create an additional sub-algorithm, denoted by $\mathsf{NPHelp}$, to make the recursion explicit. Similar to [18], our scheme also supports an arbitrary number of input encodings through extra automorphisms to satisfy Equation (1). More details can be found in Figure 3.

Unfortunately, the plaintext space of NTRU encoding limits the output encoding to only the constants of input encodings. Thus, everyone needs to calculate the rotation first to put the data into the constants.

### 3.3 Security Analysis

We provide a security analysis of NTRU packing as follows, with a full proof in Section A.1.

**Theorem 1.** *Let $\lambda$ be a security parameter and $(N, q, \chi)$ denote a parameter set of Definition 2, where the polynomial degree $N$ is a power of two, $q$ is the modulus, and $\chi$ is sub-Gaussian with parameter $\sigma_\chi$ for all errors. Define a polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and a decomposition base $B_{pk}$ with $t_{pk} = \left\lceil \log_{B_{pk}} q \right\rceil$. The NTRU packing scheme:*

- *is correct, and the error is sub-Gaussian with parameter $\hat{\sigma} \leq \sqrt{\frac{1}{36}(N^2 - 1)t_{pk} N B_{pk}^2 \sigma_\chi^2 + N^2 \sigma_\chi^2}$;*

- *satisfies pseudorandomness given the public key, if NTRU encodings satisfy Definition 5.*

# 4 The Npir Scheme

We present Npir, a novel high-rate single-server PIR scheme that integrates a unique database and a proposed packing technique. Furthermore, we introduce a batch variant, denoted by Npir$_{\mathsf{b}}$, to facilitate batch processing.

The complete construction is presented in Figure 4, along with an explicit overhead analysis. Full correctness and security analyses are provided in Section 4.2, and comprehensive experimental evaluations are provided in Section 5.

## 4.1 Construction

**Our Npir framework.** Built on NTRU encoding and our proposed NTRU packing (see overview in Figure 1), Npir operates on database $\mathbf{D} = \{d_{i,j}\}_{i\in[N\phi],j\in[\ell]} \in \mathcal{R}_p^{N\phi\times\ell}\mathbf{D} = \{d_{i,j}\}_{i\in[N\phi],j\in[\ell]} \in \mathcal{R}_p^{N\phi\times\ell}$, where each record maps to coefficients of a column-specific term (indexed by $\mathsf{ind} = (\mathsf{col}, \mathsf{term})$, $\mathsf{col} \in [\ell]$ and $\mathsf{term} \in [N]$).

In the online phase, the client generates two encodings for $\mathsf{ind} = (\mathsf{col}, \mathsf{term})$:

- one **column encoding**, which can be expanded to $\ell$ NTRU encodings via coefficient expansion; and

- one **rotation encoding**, which is used to construct a query vector via external products with expanded encodings.

Indeed, the query vector contains only the $\mathsf{col}$-th encoding with plaintext $-X^{N-\mathsf{term}}$, while the others are zero polynomials. The server then multiplies the database by query vector to obtain a vector containing $N\phi$ encodings. The constant terms in this vector comprise the targeted record. To extract the record, the server applies NTRU packing to these encodings and obtains $\phi$ NTRU encodings. After modulus switching, the server sends the response to the client, who recovers the targeted record. Note that, in Figure 4, two factors, $((2^r)^{-1} \bmod q)$ and $(N^{-1} \bmod q)$, need to be multiplied together to offset the scaling effects of expansion and packing.

**Explicit overhead.** For a database $\mathbf{D} \in \mathcal{R}_p^{N\phi\times\ell}$, we present an explicit overhead of Npir. In the offline phase, the client only needs to generate a public parameter, and the server converts the database format. Specifically,

- the client generates a public parameter of size $t_{pk}N\log_2 N\log_2 q + t_{ce}N\log_2 \ell\log_2 q$; and

- the server performs database setup (polynomial encoding + NTT conversion).

In the online phase per query, Npir requires that:

- the client generates one NTRU and one NGSW encoding and upload a query of size $N\log_2 q \cdot (1 + t_g)$;

- the server executes $\lceil\log_2 \ell\rceil$-depth query expansion, $\ell t_g$ query recovery multiplications, $N\ell\phi$ database-query multiplications, and $\phi$ packing operations; and

- the client downloads a response of size $N\phi\log_2 q_1$ and performs $\phi$ decryption.

**The PIR scheme NPIR**

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{qk})$ The algorithm does:

1. **Parameter setup:** Define the decomposition bases $B_{pk}, B_{ce}, B_g \in \mathbb{N}$ and sample an invertible secret $f \leftarrow\$\ \chi$ with $f^{-1}$.
2. **Keys generation:** Create the packing key $\mathsf{pk} \leftarrow \mathsf{NPSetup}(1^\lambda, f, B_{pk})$, and the expansion key $\mathsf{ck} \leftarrow \mathsf{CESetup}(1^\lambda, f, B_{ce}, \ell)$.
3. Return $\mathsf{pp} \leftarrow (\mathsf{pk}, \mathsf{ck}, B_{pk}, B_{ce}, B_g)$ and $\mathsf{qk} \leftarrow f$.

$\mathsf{SetupDB}(1^\lambda, \mathcal{D}) \to \mathbf{D}$ The algorithm does:

1. **Database encoding:** Define $(m, n) \leftarrow (N\phi, N\ell)$ for $\mathcal{D} = \{\mathbf{d}_i \in \mathbb{Z}_p^m\}_{i \in [n]}$, and encode $[\mathbf{d}_0 \| \cdots \| \mathbf{d}_{n-1}]$ into $\{d_{i,j}\} \in \mathcal{R}_p^{N\phi \times \ell}$ by row.
2. Return $\mathbf{D} \leftarrow \{d_{i,j}\}_{i \in [N\phi], j \in [\ell]}$.

$\mathsf{Query}(\mathsf{qk}, \mathsf{ind} = (\mathsf{col}, \mathsf{term})) \to \mathsf{qu}$ The algorithm does:

1. **Index setup:** Define a column plaintext $\mathsf{cp} \leftarrow X^{\mathsf{col}}$, a rotation plaintext $\mathsf{rp} \leftarrow -X^{N-\mathsf{term}}$, and $r \leftarrow \lceil \log_2 \ell \rceil$.
2. **Query encryption:** Generate the column encoding $\mathsf{cc} \leftarrow \mathsf{NTRU}_f(\mathsf{cp})$, and the rotation encoding $\mathsf{rc} \leftarrow \mathsf{NGSW}_f(B_g, \mathsf{rp})$.
3. Return $\mathsf{qu} \leftarrow (((2^r)^{-1} \bmod q) \cdot \mathsf{cc}, (N^{-1} \bmod q) \cdot \mathsf{rc})$.

$\mathsf{Response}(\mathsf{pp}, \mathbf{D}, \mathsf{qu}) \to \mathsf{resp}$ The algorithm does:

1. **Query recovery:** Expand $\{c_j\}_{j \in [\ell]} \leftarrow \mathsf{CoeffExpand}(\mathsf{ck}, B_{ce}, \mathsf{cc}, \ell)$, and calculate $\tilde{c}_i \leftarrow \mathsf{rc} \boxdot_{B_g} c_i$ for $i \in [\ell]$.
2. **First-dimension processing:** For $i \in [N\phi]$, compute $\hat{c}_i \leftarrow \sum_{j=0}^{\ell-1} d_{i,j} \cdot \tilde{c}_j$.
3. **Second-dimension packing:** Call $\{\rho_i\}_{i \in [\phi]} \leftarrow \mathsf{NPacking}(\mathsf{pk}, \{\hat{c}_i\})$ and switch $\{\bar{\rho}_i\}_{i \in [\phi]} \leftarrow \mathsf{Modswitch}(\{\rho_i\}_{i \in [\phi]}, q, q_1)$.
4. Return $\mathsf{resp} \leftarrow \{\bar{\rho}_i\}_{i \in [\phi]}$.

$\mathsf{Recover}(\mathsf{qk}, \mathsf{resp}) \to d$ The algorithm does:

1. **Response decoding:** For $i \in [\phi]$, compute $\mathbf{v}_i \leftarrow \mathsf{Coeff}(\bar{\rho}_i \cdot f)$, and return $d \leftarrow [\mathbf{v}_0 \| \cdots \| \mathbf{v}_{\phi-1}]^\top$.

Figure 4: The construction of our NPIR scheme.

## 4.2 Security Analysis

We present the correctness and security theorems below, along with the full proofs in Sections A.2 and A.3.

**Theorem 2.** *Given a security parameter $\lambda$ and a database $\mathbf{D} \in \mathcal{R}_p^{N\phi \times \ell}$, suppose:*

- *$(N, q, \chi)$ denote a parameter set of Definition 2, where the polynomial degree $N$ is a power of two, the modulus $q$ is the product of two primes $q_1$ and $q_2$, and $\chi$ is sub-Gaussian with* *a parameter of $\sigma_\chi$ for all errors;*

- *The three decomposition parameter sets are respectively given by: $(t_{pk}, B_{pk}), (t_{ce}, B_{ce})$ and $(t_g, B_g)$;*
- *$p$ is the plaintext modulus; $\Delta = \lfloor q/p \rfloor$ and $\Delta_1 = \lfloor q_1/p \rfloor$.*

*Suppose that the parameter set satisfies Equation* (2)*. Then, the* Npir *scheme is $\delta$-correct.*

**Theorem 3** (Query privacy for $\mathcal{Q}$ queries)**.** *Given a security parameter $\lambda$ and $\mathcal{Q}$ queries with an index set $\mathcal{I}$, Definitions 2 and 5 hold for the parameter set. Then, the* Npir *scheme satisfies query privacy for $\mathcal{Q}$ queries.*

## 4.3 Extensions

We present several extensions to Npir, including support for batch query processing and variable record sizes.

### 4.3.1 Batch processing

We extend Npir to a batch variant Npir$_\mathsf{b}$ for simultaneous multi-record queries. We omit batch codes [3,32] to reduce database traversal for implementation simplicity. Leveraging unused capacity in column plaintexts ($\ell{-}1 \le N{-}1$), the client groups $T$ batch indices into subsets of size $\lfloor N/\ell \rfloor$ and encodes them into $\lceil T\ell/N \rceil$ column encodings and $T$ NGSW encodings. The server runs $\lceil T\ell/N \rceil$ **Query recovery** instances, as well as $T$ retrievals (including **First-dimension processing** and **Second-dimension packing**). The response plaintext space is fully utilized, and no extra compression is needed.

### 4.3.2 Varying record sizes

Our scheme naturally accommodates records of arbitrary lengths. As described in Figure 1, we organize each record into polynomial coefficients within a column.By modifying the row count and packing range, we can adjust record size. Unlike the iterative query approach in [43,52,53], for larger records we simply add rows and perform additional packing operations.

# 5 Implementation and Evaluation

In this section, we present a Rust implementation of Npir, comparing it with recent advanced schemes. Our code is available at `https://github.com/llllinyl/npir`.

## 5.1 Experimental Setup and Parameters

**Implementation details.** We implement NTRU packing, Npir, and Npir$_\mathsf{b}$ in approximately 4,600 lines of Rust and 50 lines of C++. We use the NTL library[2] and the NTT module from Spiral-rs[3] [43]. Experiments are conducted on an Aliyun ECS.r7.16xlarge instance (Ubuntu 22.04) with 64 vCPUs (Intel Xeon Ice Lake Platinum 8369B at 2.70 GHz) and 512 GB of RAM, leveraging AVX2 for SIMD optimizations.
**Comparison baselines.** We compare against the following: OnionPIR [45], Spiral[3] [43], CwPIR [41], VBPIR [46], NTRUPIR [53] (reproduced due to an absence of a public implementation), Respire [13], KsPIR [40], and PIRANA [38]. LightPIR [52] (which lacks a public

[2] NTL-11.5.1, available at `https://libntl.org`.

[3] `https://github.com/menonsamir/spiral-rs`.

Table 1: A comparison of retrieving a single moderate-size record (32 KB per record).

| Database | Metric | OnionPIR† | Spiral | NTRUPIR | CwPIR | PIRANA | Npir |
|---|---|---|---|---|---|---|---|
| | Rate | 0.250 | 0.390 | 0.444 | 0.155 | 0.153 | 0.250 |
| 1 GB<br>$2^{15}$×32 KB | Storage (MB) | 5 | 8.38 | 6.13 | 14.83 | 5.52 | 0.89 |
| | Preproc. (s) | 22.31 | 29.86 | 13.32 | 19.84 | 42.15 | 13.32 |
| | Query (KB) | 64 | 16 | 24 | 216 | 590.97 | 84 |
| | Response (KB) | 128 | 82 | 72 | 206 | 209.07 | 128 |
| | Server time (s) | 8.33 | 8.76 | 10.39 | 540.75 | 8.61 | 5.84 |
| | Throughput (MB/s) | 122.93 | 116.89 | 98.56 | 1.89 | 118.93 | 175.34 |
| | Client time (ms) | 4.09 | 2.34 | 7.13 | 26.34 | 11.75 | 1.61 |
| 4 GB<br>$2^{17}$×32 KB | Storage (MB) | 5 | 8.50 | 6.13 | 14.83 | 5.52 | 1.11 |
| | Preproc. (s) | 90.71 | 128.82 | 52.47 | 79.46 | 179.29 | 52.47 |
| | Query (KB) | 64 | 16 | 24 | 216 | 1036.41 | 84 |
| | Response (KB) | 128 | 82 | 72 | 206 | 209.02 | 128 |
| | Server time (s) | 29.33 | 21.22 | 25.04 | 2163.34 | 32.11 | 9.79 |
| | Throughput (MB/s) | 139.65 | 193.03 | 163.58 | 1.89 | 127.56 | 418.39 |
| | Client time (ms) | 4.30 | 2.38 | 7.54 | 26.58 | 18.89 | 1.61 |
| 8 GB<br>$2^{18}$×32 KB | Storage (MB) | 5 | 8.63 | 6.13 | ‡ | 5.52 | 1.22 |
| | Preproc. (s) | 181.54 | 268.14 | 111.06 | ‡ | 392.83 | 111.06 |
| | Query (KB) | 64 | 16 | 24 | ‡ | 1330.95 | 84 |
| | Response (KB) | 128 | 82 | 72 | ‡ | 209.04 | 128 |
| | Server time (s) | 56.98 | 35.47 | 36.78 | ‡ | 63.19 | 14.87 |
| | Throughput (MB/s) | 143.77 | 230.96 | 222.73 | ‡ | 129.64 | 550.91 |
| | Client time (ms) | 4.27 | 2.36 | 7.51 | ‡ | 23.19 | 1.62 |
| 16 GB<br>$2^{19}$×32 KB | Storage (MB) | 5 | 8.63 | 6.13 | ‡ | 5.52 | 1.33 |
| | Preproc. (s) | 365.26 | 559.53 | 222.79 | ‡ | 830.97 | 222.79 |
| | Query (KB) | 64 | 16 | 24 | ‡ | 1774.98 | 84 |
| | Response (KB) | 128 | 82 | 72 | ‡ | 208.99 | 128 |
| | Server time (s) | 112.98 | 70.43 | 61.56 | ‡ | 124.52 | 24.76 |
| | Throughput (MB/s) | 145.02 | 232.63 | 266.15 | ‡ | 131.58 | 661.71 |
| | Client time (ms) | 4.45 | 2.34 | 7.48 | ‡ | 31.13 | 1.61 |
| 32 GB<br>$2^{20}$×32 KB | Storage (MB) | 5 | 8.75 | 6.13 | ‡ | 5.52 | 1.44 |
| | Preproc. (s) | 752.51 | 1153.31 | 437.69 | ‡ | 1704.65 | 437.69 |
| | Query (KB) | 64 | 16 | 24 | ‡ | 2511.02 | 84 |
| | Response (KB) | 128 | 82 | 72 | ‡ | 209.05 | 128 |
| | Server time (s) | 228.41 | 120.42 | 105.13 | ‡ | 245.32 | 45.82 |
| | Throughput (MB/s) | 143.46 | 272.11 | 311.69 | ‡ | 133.57 | 715.15 |
| | Client time (ms) | 4.46 | 2.29 | 7.57 | ‡ | 44.24 | 1.61 |

† OnionPIR does not support 32 KB, so we simulated it with 30 KB records. The public implementation of OnionPIR does not provide the size of the public parameter, so we refer to Table 2 in [43].

‡ Our environment cannot support the RAM usage required by CwPIR when the database exceeds 4 GB.

**Storage** refers to the space needed to store the public parameter stored by the server, and **Preproc.** is the time taken to prepare the database. **Server time** is the online response time , and **Throughput** is the rate between dataset scale and server time. **Client time** is the computational overhead for query generation and record recovery. This concept is further repeated in Tables 2 to 5.

implementation) can be seen as a conversion of Spiral from RLWE to RLWR, so it is not reproduced for comparison. All experiments are run in a single thread, and results are averaged over five sample runs. Communication is measured in aligned bytes, with protocol interactions being ignored through local simulation.

In evaluating NPIR, we first address the primary research question (RQ) related to the main focus of our work. This is followed by two supplementary RQs that explore component-level behavior and cross-scenario applicability:

- **Primary RQ**: How does the performance of NPIR compare to that of existing high-rate PIRs, particularly the state-of-the-art baselines Spiral and NTRUPIR, for moderate-size record retrieval?
- **Supplementary RQ1**: How does each independent component of NPIR perform, and how does the protocol's overall performance scale with record size?
- **Supplementary RQ2**: How does NPIR perform in extended practical scenarios, including batch query processing and retrieval of small and large records?

**Parameter selection.** Following Theorems 1 and 2, we select a parameter set that achieves a correctness error of at most $2^{-40}$ while maintaining NIST-I security via the lattice estimator[4] in [1]. More details are provided below.

- We first set the NTRU parameter set: dimension $N = 2048$, modulus $q \approx 2^{54}$, and error distribution $\chi$, which is a sub-Gaussian distribution with parameter $\sigma_\chi = \sqrt{\frac{4\pi}{3}}$.
- We employ the fast number-theoretic transform (NTT) technique [39] to accelerate polynomial multiplications. To do so, we use a modulus $q$, represented as a product of two NTT-friendly primes, that is, $q = q_1 \cdot q_2$ where $q_1 = 11 \cdot 2^{21} + 1$ and $q_2 = 479 \cdot 2^{21} + 1$.
- Our schemes involve three types of gadgets: (i) $(t_{pk}, B_{pk}) = (3, 2^{19})$ for NTRU packing; (ii) $(t_{ce}, B_{ce}) = (8, 2^7)$ for coefficient expansion; and (iii) $(t_g, B_g) = (5, 2^{11})$ for NGSW encoding.
- We choose the plaintext modulus $p = 256$, and in the dimension $(N\phi, N\ell)$ of the initial database, $\phi$ is the packing number and $\ell$ is the column number of preprocessed database. The record size is equal to the packing number multiplied by the modulus (i.e., $N\phi \log_2 p$ bits). Therefore, we define the packing number $\phi$ as 16 for 32 KB in Tables 1 and 3, 1 for 2 KB in Table 4, and 64 for 128 KB in Table 5.

## 5.2 Evaluation for Moderate-size Records

As a core contribution to answering **Primary RQ**, we evaluate NPIR against high-rate schemes in Table 1 and Figure 5 that provide concrete evaluations for moderate-size records. These include OnionPIR [45], Spiral [43], and NTRUPIR [53]. We also include the constant-weight PIRs CwPIR [41] and PIRANA [38], which provide evaluations for moderate-size records and offer high communication rates. KsPIR [40] is excluded because its public implementation only supports 8 KB records; simply expanding to 32 KB (multiplying the time by four) would not provide a fair comparison.

**Test setup:** All tests use 32 KB records, a binary-friendly size. The best performance metrics are highlighted in green. Note that NTRUPIR and NPIR share identical preprocessing overheads due to NTT database conversion.

[4] https://github.com/malb/lattice-estimator

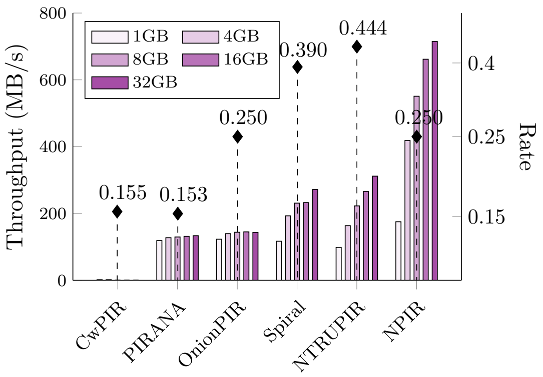

Figure 5: Throughput versus rate for varying databases with moderate-size records.

Npir demonstrates significant advantages across multiple metrics in Table 1. Our solution has the fastest preprocessing time, identical to NTRUPIR's, and the smallest server storage size. Client-side computation time is also optimal with reductions ranging from 1.45 to 16.51 times compared to other schemes. In terms of server processing time, Npir outperforms Spiral by 1.50 to 2.84 times and NTRUPIR by 1.77 to 2.55 times for databases ranging from 1 GB to 32 GB. We attribute the improvement in server time to the following reasons:

- Npir requires only $\lceil \log_2 \ell \rceil$ depth expansion, whereas Spiral and NTRUPIR require additional expansions related to the second dimension for folding;
- and Npir adopts a novel NTRU packing method to extract all constant terms, rather than using the homomorphic selection method in Spiral and NTRUPIR.

Due to techniques such as coefficient expansion, homomorphic selection, and packing, the communication overhead for all schemes except PIRANA is a constant number of polynomials. Unfortunately, Npir does not have any advantages in terms of query or response size: the query size in Npir is 5.25 times larger than Spiral's, and the response size is the same as OnionPIR's but 1.78 times larger than Spiral's.

## 5.3 Microbenchmarks

This section provides a detailed analysis of the server runtime in Table 1 and the effect that different record sizes have on performance, in order to answer **Supplementary RQ1**.

Table 2: The breakdown of Npir.

| | Metric | 4 GB | 8 GB | 16 GB | 32 GB |
|---|---|---|---|---|---|
| Communication (KB) | *Query* | 84 | | | |
| | *Response* | 128 | | | |
| Client time (ms) | *Query* | 0.54 | 0.57 | 0.56 | 0.55 |
| | *Recovery* | 1.07 | 1.05 | 1.05 | 1.06 |
| Server offline time (s) | | 52.47 | 111.06 | 222.79 | 437.69 |
| Server time (s) | *Expand* | 0.04 | 0.08 | 0.17 | 0.35 |
| | *Multiply* | 5.14 | 10.18 | 19.98 | 40.86 |
| | *Packing* | 4.61 (0.288 amortized) | | | |
| | Total | 9.79 | 14.87 | 24.76 | 45.82 |

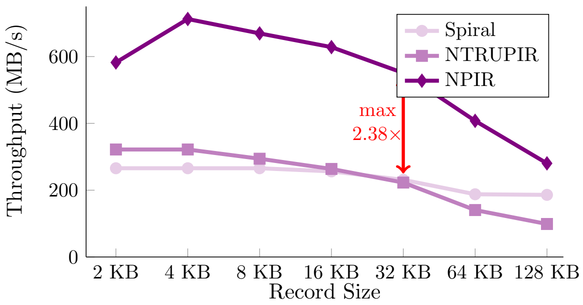

Figure 6: Comparison against Spiral and NTRUPIR with varying record sizes.

### 5.3.1 Microbenchmarks

Table 2 provides a detailed breakdown of NPIR in Table 1, identifying three main phases of the server's online time: *Expand*, *Multiply*, and *Packing*. Due to expansion and NTRU packing, NPIR has constant communication overhead (including queries and responses) and client time (including query generation for two encodings and recovery for $\phi$ encodings). Server preprocessing scales linearly with database size. Packing overhead is fixed and amortized at 0.288 seconds per operation for server time. Expansion time increases linearly with the number of database columns. However, the main cost of server time arises from the calculation of the product of the database and query vectors, especially for large databases.

### 5.3.2 Varying record sizes

We evaluate the throughput of NPIR across different record sizes and compare it to two baselines: Spiral [43] and NTRUPIR [53] (see Figure 6). All experiments use an 8 GB database, with record sizes ranging from 2 KB to 128 KB. Overall, NPIR maintains a leading position, achieving a maximum optimization ratio of 2.38 times at 32 KB. In NPIR, an increase in record size affects two aspects: **reduced** query ciphertext recovery required for first-dimension processing and **increased** packing frequency for the second dimension. Due to these two factors, our work exhibits a non-monotonic trend: the optimal throughput is achieved at a record size of 4 KB.

## 5.4 Extension Experiments

In order to answer **Supplementary RQ2**, this section provides further validation of NPIR's applicability through additional scenarios, including batch query processing and performance across diverse records.

### 5.4.1 Case study i: applicability of batch queries

Table 3 compares the batch variant NPIR$_\mathsf{b}$ with VBPIR [46] and PIRANA [38], which support processing batches of moderate-size records. However, we exclude OnionPIR [45], Spiral [43], and NTRUPIR [53] from batch comparisons, since their public implementations lack batch support.

**Test setup:** All tests use 32 KB records for consistency, and the best performance is highlighted in green. We conduct the experiments across two scenarios with different batch sizes and dataset scales.

NPIR$_\mathsf{b}$ shows advantages in offline metrics (similar to Table 1), response size, and client time. Specifically, the response size is 6.15 to 521.23 times smaller in all tests, and client time improves with speedups of 3.29 to 678.39 times. The query size is larger than the others because

Table 3: A comparison of processing batch queries.

| Database | Metric | VBPIR | PIRANA | NPIR$_\mathsf{b}$ | VBPIR | PIRANA | NPIR$_\mathsf{b}$ |
|---|---|---|---|---|---|---|---|
| | Batch Size | 8 | | | 32 | | |
| | Storage (MB) | 9.3 | 5.52 | 1.65 | 9.3 | 5.52 | 1.65 |
| 1 GB<br>$2^{15}$×32 KB | Preproc. (s) | 512.01 | 293.74 | 13.32 | 793.65 | 296.02 | 13.32 |
| | Query (KB) | 385 | 502.56 | 574 | 385 | 507.87 | 2254 |
| | Response (KB) | 6427 | 533746 | 1024 | 25196 | 533755 | 4096 |
| | Server time (s) | 259.92 | 65.59 | 47.19 | 254.71 | 65.56 | 191.24 |
| | Client time (ms) | 42.71 | 7767.53 | 11.45 | 150.39 | 7845.74 | 45.73 |
| 8 GB<br>$2^{18}$×32 KB | Preproc. (s) | ‡ | 2207.29 | 111.06 | ‡ | 2261.18 | 111.06 |
| | Query (KB) | ‡ | 1061.91 | 574 | ‡ | 1065.95 | 2286 |
| | Response (KB) | ‡ | 533748 | 1024 | ‡ | 533755 | 4096 |
| | Server time (s) | ‡ | 313.56 | 126.85 | ‡ | 310.72 | 469.85 |
| | Client time (ms) | ‡ | 7822.86 | 11.59 | ‡ | 7782.59 | 45.21 |

‡ Our environment cannot support the required RAM usage for VBPIR when the database exceeds 8 GB. The public implementation of VBPIR does not provide the size of the public parameter, so we refer to Table 2 in [13].

NGSW encodings in NPIR cannot be compressed. However, the overall communication overhead improves by 4.26 to 334.67 times. Our solution differs from VBPIR and PIRANA in that it increases server time linearly with batch size. For 8-query batches, performance is 1.39 to 5.51 times faster. However, with larger batch sizes, NPIR$_\mathsf{b}$ becomes less efficient than PIRANA for two reasons. First, there are no batch codes [3, 32] that reduce database traversal. Second, there are SIMD/constant-weight optimizations employed in [38, 46].

Table 4: A comparison of retrieving a single small record.

| Database | Metric | OnionPIR<br>(4 KB) | Spiral<br>(8 KB) | NTRUPIR<br>(4 KB) | Respire<br>(256 B) | KsPIR<br>(8 KB) | NPIR<br>(2 KB) |
|---|---|---|---|---|---|---|---|
| 1 GB | Storage (MB) | 5 | 7.88 | 6.13 | 3.89 | 8.8 | 1.33 |
| | Preproc. (s) | 27.98 | 34.73 | 13.32 | 71.57 | 13.60 | 13.32 |
| | Query (KB) | 64 | 16 | 24 | 7.66 | 140 | 84 |
| | Response (KB) | 128 | 20.5 | 9 | 2 | 26 | 8 |
| | Server time (s) | 9.93 | 5.94 | 4.62 | 3.16 | 0.72 | 1.72 |
| | Throughput (MB/s) | 103.12 | 172.39 | 221.65 | 324.05 | 1422.22 | 595.35 |
| | Client time (ms) | 4.19 | 1.39 | 0.93 | 0.41 | 5.79 | 0.59 |
| 8 GB | Storage (MB) | 5 | 8.75 | 6.13 | 3.89 | 8.8 | 1.65 |
| | Preproc. (s) | 230.95 | 286.17 | 111.06 | 577.61 | 113.76 | 111.06 |
| | Query (KB) | 64 | 16 | 24 | 14.77 | 140 | 84 |
| | Response (KB) | 128 | 20.5 | 9 | 2 | 26 | 8 |
| | Server time (s) | 72.43 | 30.84 | 25.46 | 16.54 | 5.37 | 14.07 |
| | Throughput (MB/s) | 113.10 | 265.63 | 321.76 | 495.28 | 1525.51 | 582.23 |
| | Client time (ms) | 4.36 | 1.36 | 0.97 | 0.43 | 5.90 | 0.58 |

### 5.4.2 Case study ii: applicability of small records

Table 4 compares Npir with high-rate PIR schemes for retrieving a single small record retrieval, including OnionPIR [45], Spiral [43], NTRUPIR [53], Respire [13], and KsPIR [40] under their own default parameters.

**Test setup:** Each scheme operates at **its optimal and most recommended record size** (shown below each solution name in Table 4). This setup follows the evaluation approach in [40], which ensures a fair comparison. The top two performances per metric are highlighted in green.

Npir exhibits advantages in offline performance (similar to Table 1). For online metrics, Npir achieves the following performance: response size (second only to Respire), server time (second only to KsPIR), and client time (second only to Respire). These differences stem from the BSGS algorithm used in KsPIR and the subring structure deployed in Respire. Additionally, the query size remains 1.31–10.97 times larger than that of other PIRs, except for KsPIR.

Table 5: A comparison of retrieving a single large record.

| Database | Metric | Spiral | NTRUPIR | Npir |
|---|---|---|---|---|
| 1 GB | Storage (MB) | 8.06 | 6.13 | 0.67 |
| | Preproc. (s) | 28.79 | 13.35 | 13.35 |
| | Query (KB) | 16 | 24 | 84 |
| | Response (KB) | 328 | 288 | 512 |
| | Server time (s) | 6.08 | 11.64 | 20.82 |
| | Throughput (MB/s) | 168.42 | 87.97 | 49.18 |
| | Client time (ms) | 5.37 | 8.14 | 4.67 |
| 8 GB | Sto rage (MB) | 8.94 | 6.13 | 1.00 |
| | Preproc. (s) | 272.56 | 106.51 | 106.51 |
| | Query (KB) | 16 | 24 | 84 |
| | Response (KB) | 328 | 288 | 512 |
| | Server time (s) | 44.06 | 34.59 | 29.23 |
| | Throughput (MB/s) | 185.92 | 236.83 | 280.26 |
| | Client time (ms) | 5.43 | 8.34 | 4.61 |

### 5.4.3 Case study iii: applicability of large records

We further compare Npir with high-rate PIRs (Spiral [43] and NTRUPIR [53]), which provide concrete evaluations for large records (>100 KB).

**Test setup:** Experiments are conducted on 1 GB ($2^{13}\times$128 KB) and 8 GB ($2^{16}\times$128 KB) databases containing 128 KB records. The best performance results are highlighted in green.

For a 1 GB database, Npir only outperforms the baselines in terms of offline metrics and client time. Server time is limited by the 64 times NTRU packing operations required for 128 KB records. However, for an 8 GB database, Npir is faster in terms of server time. For instance, it is 1.57 times faster than Spiral and 1.18 times faster than NTRUPIR. This is because our scheme only affects query recovery and database-query multiplication when scaling the database, and packing overhead depends solely on record size and remains constant for the same record size. In contrast, Spiral and NTRUPIR incur additional overhead in second-dimension folding.

# 6 Related Work

Chor et al. first introduced private information retrieval (PIR) in 1995 [19] as a multi-server framework that requires non-colluding servers to safeguard user privacy. Subsequent multi-server PIRs either rely on computational assumptions [9,12,26,28] or on lightweight information theory [6–8, 24, 51, 55]. However, their dependence on non-colluding servers remains a critical limitation.

**Single-server PIRs.** To address this limitation, Kushilevitz and Ostrovsky [36] pioneered single-server PIR, laying the foundation for modern research. This work has driven extensive research focusing on optimizing the offline/online performance of single-server PIRs. Recently, fully homomorphic encryption (FHE)-based schemes, such as [2,3,13–15,20,21,25,31,37,38,40–47,52,53], have gained prominence recently due to their concrete efficiency. Recent FHE-based single-server PIRs fall into two key categories based on distinct optimization objectives: (1) high-throughput schemes and (2) high-rate schemes.

**High-throughput PIRs.** High-throughput schemes (e.g., [21, 30, 31, 37, 44]) focus on accelerating server-side online computation via offline preprocessing (e.g., large hint generation in [21,31]). Subsequent solutions [30,37,44] eliminate hint transmission using bootstrapping or packing, yet still require costly preprocessing.

**High-rate PIRs.** This lineage focuses on bandwidth efficiency for constrained networks and includes [13,40,43,45,52,53]. Initially, OnionPIR [45] began researching how to implement such a PIR scheme for datasets with moderate-size records based on SealPIR [3]. Subsequently, [43] formally defined the concept of *rate* to evaluate server-to-client communication and proposed the Spiral family, which involves records of various lengths. Works in [52,53] targeted the Spiral family and introduced the NTRU lattice and the ring learning with rounding (RLWR) problem, respectively, to improve performance. Works such as [13,40] further improved performance for databases with small records, achieving breakthroughs in communication and server computing, respectively.

**Functional extensions.** In practical scenarios, most databases use key-value storage, prompting solutions such as [2,16,29,41,48] to extend traditional PIRs into keyword PIRs for key-based retrieval. To mitigate malicious tampering, works such as [10, 14, 20, 22, 56] focus on verifiable PIR designs that ensure record integrity via cryptographic mechanisms. A core functional extension is batch PIR [32], which enables multi-record retrieval in one query. Schemes such as [3, 11, 38, 41, 46] support this efficiently via vectorized techniques (e.g., SIMD and constant-weight codes) for compact encoding and server-side processing.

# 7 Conclusion and Future Work

We propose Npir, a high-rate, single-server PIR scheme based on NTRU that integrates a novel database with an efficient NTRU packing technique. This design provides the highest throughput for moderate-size records compared to other high-rate PIR schemes and is applicable to batch queries and variable-record scenarios. Notably, our NTRU packing technique serves as a compression mechanism, preserving only the constant terms when merging multiple NTRU encodings.

Future work will include exploring NGSW encoding compression, batch code integration, and parameter optimization following [23], as well as extending this framework to other NTRU-based privacy-enhancing protocols.

# References

[1] Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. pp. 169–203 (2015)

[2] Ali, A., Lepoint, T., Patel, S., Raykova, M., Schoppmann, P., Seth, K., Yeo, K.: Communication-computation trade-offs in PIR. In: 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021. pp. 1811–1828. USENIX Association (2021)

[3] Angel, S., Chen, H., Laine, K., Setty, S.T.V.: PIR with compressed queries and amortized query processing. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 962–979. IEEE (2018)

[4] Angel, S., Setty, S.T.V.: Unobservable communication over fully untrusted infrastructure. In: 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016. pp. 551–569. USENIX Association (2016)

[5] Backes, M., Kate, A., Maffei, M., Pecina, K.: Obliviad: Provably secure and practical online behavioral advertising. In: IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA. pp. 257–271. IEEE Computer Society (2012)

[6] Beimel, A., Ishai, Y., Kushilevitz, E.: General constructions for information-theoretic private information retrieval. J. Comput. Syst. Sci. pp. 213–247 (2005)

[7] Beimel, A., Ishai, Y., Kushilevitz, E., Orlov, I.: Share conversion and private information retrieval. In: Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012. pp. 258–268. IEEE (2012)

[8] Beimel, A., Ishai, Y., Kushilevitz, E., Raymond, J.: Breaking the o(n1/(2k-1)) barrier for information-theoretic private information retrieval. In: 43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada. pp. 261–270. IEEE (2002)

[9] Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers computation in private information retrieval: PIR with preprocessing. In: Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000. pp. 55–73. Springer (2000)

[10] Ben-David, S., Kalai, Y.T., Paneth, O.: Verifiable private information retrieval. In: Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022. pp. 3–32. Springer (2022)

[11] Bienstock, A., Patel, S., Seo, J.Y., Yeo, K.: Batch PIR and labeled PSI with oblivious ciphertext compression. In: 33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024. USENIX Association (2024)

[12] Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 1292–1303. ACM (2016)

[13] Burton, A., Menon, S.J., Wu, D.J.: Respire: High-rate PIR for databases with small records. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 1463–1477. ACM (2024)

[14] de Castro, L., Lee, K.: Verisimplepir: Verifiability in simplepir at no online cost for honest servers. In: 33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024. USENIX Association (2024)

[15] de Castro, L., Lewi, K., Suh, E.: Whispir: Stateless private information retrieval with low communication. IACR Cryptol. ePrint Arch. p. 266 (2024)

[16] Celi, S., Davidson, A.: Call me by my name: Simple, practical private information retrieval for keyword queries. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 4107–4121. ACM (2024)

[17] Chen, H., Chillotti, I., Ren, L.: Onion ring ORAM: efficient constant bandwidth oblivious RAM from (leveled) TFHE. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019. pp. 345–360. ACM (2019)

[18] Chen, H., Dai, W., Kim, M., Song, Y.: Efficient homomorphic conversion between (ring) LWE ciphertexts. In: Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021. pp. 460–479. Springer (2021)

[19] Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM pp. 965–981 (1998)

[20] Colombo, S., Nikitin, K., Corrigan-Gibbs, H., Wu, D.J., Ford, B.: Authenticated private information retrieval. In: 32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023. pp. 3835–3851. USENIX Association (2023)

[21] Davidson, A., Pestana, G., Celi, S.: Frodopir: Simple, scalable, single-server private information retrieval. Proc. Priv. Enhancing Technol. pp. 365–383 (2023)

[22] Dietz, M., Tessaro, S.: Fully malicious authenticated PIR. In: Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024. pp. 113–147. Springer (2024)

[23] Ducas, L., van Woerden, W.P.J.: NTRU fatigue: How stretched is overstretched? In: Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021. pp. 3–32. Springer (2021)

[24] Efremenko, K.: 3-query locally decodable codes of subexponential length. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 39–44. ACM (2009)

[25] Gentry, C., Halevi, S.: Compressible FHE with applications to PIR. In: Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019. pp. 438–464. Springer (2019)

[26] Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. pp. 640–658. Springer (2014)

[27] Green, M., Ladd, W., Miers, I.: A protocol for privately reporting ad impressions at scale. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 1591–1601. ACM (2016)

[28] Hafiz, S.M., Henry, R.: A bit more than a bit is more than a bit better: Faster (essentially) optimal-rate many-server PIR. Proc. Priv. Enhancing Technol. pp. 112–131 (2019)

[29] Hao, M., Liu, W., Peng, L., Zhang, C., Wu, P., Zhang, L., Li, H., Deng, R.H.: Practical keyword private information retrieval from key-to-index mappings. In: 34rd USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025. USENIX Association (2025)

[30] Henzinger, A., Dauterman, E., Corrigan-Gibbs, H., Zeldovich, N.: Private web search with tiptoe. In: Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023. pp. 396–416. ACM (2023)

[31] Henzinger, A., Hong, M.M., Corrigan-Gibbs, H., Meiklejohn, S., Vaikuntanathan, V.: One server for the price of two: Simple and fast single-server private information retrieval. In: 32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023. pp. 3889–3905. USENIX Association (2023)

[32] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Batch codes and their applications. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004. pp. 262–271. ACM (2004)

[33] Juels, A.: Targeted advertising ... and privacy too. In: Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001. pp. 408–424. Springer (2001)

[34] Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: 28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019. pp. 1447–1464. USENIX Association (2019)

[35] Kirchner, P., Fouque, P.: Revisiting lattice attacks on overstretched NTRU parameters. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017. pp. 3–26. Springer (2017)

[36] Kushilevitz, E., Ostrovsky, R.: Replication is NOT needed: SINGLE database, computationally-private information retrieval. In: 38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997. pp. 364–373. IEEE (1997)

[37] Li, B., Micciancio, D., Raykova, M., Schultz, M.: Hintless single-server private information retrieval. In: Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024. pp. 183–217. Springer (2024)

[38] Liu, J., Li, J., Wu, D., Ren, K.: PIRANA: faster multi-query PIR via constant-weight codes. In: IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024. pp. 4315–4330. IEEE (2024)

[39] Longa, P., Naehrig, M.: Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In: Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings. pp. 124–139. Springer (2016)

[40] Luo, M., Liu, F., Wang, H.: Faster fhe-based single-server private information retrieval. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 1405–1419. ACM (2024)

[41] Mahdavi, R.A., Kerschbaum, F.: Constant-weight PIR: single-round keyword PIR via constant-weight equality operators. In: 31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022. pp. 1723–1740. USENIX Association (2022)

[42] Melchor, C.A., Barrier, J., Fousse, L., Killijian, M.: XPIR : Private information retrieval for everyone. Proc. Priv. Enhancing Technol. pp. 155–174 (2016)

[43] Menon, S.J., Wu, D.J.: SPIRAL: fast, high-rate single-server PIR via FHE composition. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022. pp. 930–947. IEEE (2022)

[44] Menon, S.J., Wu, D.J.: YPIR: high-throughput single-server PIR with silent preprocessing. In: 33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024. pp. 5985–6002. USENIX Association (2024)

[45] Mughees, M.H., Chen, H., Ren, L.: Onionpir: Response efficient single-server PIR. In: CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021. pp. 2292–2306. ACM (2021)

[46] Mughees, M.H., Ren, L.: Vectorized batch private information retrieval. In: 44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023. pp. 437–452. IEEE (2023)

[47] Park, J., Tibouchi, M.: SHECS-PIR: somewhat homomorphic encryption-based compact and scalable private information retrieval. In: Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020. pp. 86–106. Springer (2020)

[48] Patel, S., Seo, J.Y., Yeo, K.: Don't be dense: Efficient keyword PIR for sparse databases. In: 32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023. pp. 3853–3870. USENIX Association (2023)

[49] Peikert, C.: A decade of lattice cryptography. Found. Trends Theor. Comput. Sci. pp. 283–424 (2016)

[50] Ren, L., Mughees, M.H., Sun, I.: Simple and practical amortized sublinear private information retrieval using dummy subsets. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 1420–1433. ACM (2024)

[51] Woodruff, D.P., Yekhanin, S.: A geometric approach to information-theoretic private information retrieval. In: 20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA. pp. 275–284. IEEE (2005)

[52] Xia, H., Wang, M.: Lightpir: Single-server PIR via FHE without gaussian noise. In: Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2024, Singapore, July 1-5, 2024. ACM (2024)

[53] Xia, H., Wang, M.: Single-server PIR via ntru-based FHE: simpler, smaller, and faster. In: 9th IEEE European Symposium on Security and Privacy, EuroS&P 2024, Vienna, Austria, July 8-12, 2024. pp. 293–310. IEEE (2024)

[54] Xiang, B., Zhang, J., Deng, Y., Dai, Y., Feng, D.: Fast blind rotation for bootstrapping fhes. In: Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023. pp. 3–36. Springer (2023)

[55] Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. J. ACM pp. 1–16 (2008)

[56] Zhao, L., Wang, X., Huang, X.: Verifiable single-server private information retrieval from LWE with binary errors. Inf. Sci. pp. 897–923 (2021)

[57] Zhou, M., Lin, W., Tselekounis, Y., Shi, E.: Optimal single-server private information retrieval. In: Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023. pp. 395–425. Springer (2023)

# A Appendix

## A.1 Proof of Theorem 1

*Proof of Correctness in Theorem 1.* We analyze correctness and error distribution throughout the packing process.

Suppose that a set of NTRU encodings is $\mathbf{C} = [c_0 \mid c_1 \mid \cdots c_{N-1}] \in \mathcal{R}_q^N$ where $c_i \cdot f = v_i + \gamma_i(X)$ and $\gamma_i(X)$ represents the non-constant terms of $c_i \cdot f$. Furthermore, $v_i$ equals the constant term in the underlying plaintext with an error.

Based on the NTRU automorphism and the correctness of the LWEs-to-RLWE ring packing described in [18], we discuss recursive calls. When $\eta = 1$, only the value of the corresponding encoding is returned. When $\eta = 2$, for $i \in [N/2]$ we can obtain the following

$$\frac{\cdots + v_i + X^{N/2} \cdot v_{i+N/2} + \tau_3(v_i + X^{N/2} \cdot v_{i+N/2})}{f},$$

where we ignore $\gamma_i(X)$ for simplicity. Similarly, we can perform calculations for each level in Figure 3. Finally, at the level $\eta = \log_2 N$, according to [18], we can obtain

$$\hat{c} = \frac{\cdots + \sum_{j\in[N]} X^j \cdot \sum_{i\in[N]} \tau_{2i+1}(v_j)}{f}.$$

For $i \in [N]$, the term $\gamma_i(X)$ will be eliminated by Equation (1) and the result will be equivalent to

$$\hat{c} = \frac{\cdots + N(v_0 + v_1 \cdot X + v_2 \cdot X^2 + \cdots + v_{N-1} \cdot X^{N-1})}{f}.$$

Therefore, when ignoring the error, the result satisfies the equation $\hat{c} \cdot f \approx N \cdot \sum_{i\in[N]} v_i \cdot X^i$, which proves its correctness.

Next, we analyze the errors generated in NTRU packing:

- The first part comes from the set of input NTRU encodings. Specifically, the value $v_i$ consists of plaintext and an error at a given position. This part is similarly amplified by a factor of $N$ in Equation (1).

- Each automorphism introduces the second part of the error. This error comes from the external product, that is, the calculation of $\mathbf{W}_\eta \boxdot_{B_{pk}} \tau_{2^\eta+1}(\beta)$ in the $\mathsf{NPHelp}$ algorithm. Influenced by $t_{pk}$ and $B_{pk}$, the additional error of this part is parameterized by the quantity $\sigma_1 = \sqrt{\frac{1}{12} t_{pk} N B_{pk}^2 \sigma_\chi^2}$. According to the conclusion in [18], this will be reduced to a parameter of $\sigma_2 \le \sqrt{\frac{1}{3}(N^2-1)\sigma_1^2} = \sqrt{\frac{1}{36}(N^2-1) t_{pk} N B_{pk}^2 \sigma_\chi^2}$ after the full packing process.

In summary, the final error is a sub-Gaussian distribution with a parameter of $\hat{\sigma} := \sqrt{\sigma_2^2 + N^2\sigma_\chi^2} \le \sqrt{\frac{1}{36}(N^2-1) t_{pk} N B_{pk}^2 \sigma_\chi^2 + N^2\sigma_\chi^2}$. □

Building on the concept of circular security from [43,44], we define the same property based on Definition 2. Circular security is also referred to as *key-dependent pseudorandomness* for NTRU encodings. The formal definition is below.

**Definition 5** (Key-dependent pseudorandomness for NTRU encodings)**.** *Let $\lambda$ be a security parameter, $(N, q, \chi)$ denote a parameter set of Definition 2, $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$ be a polynomial ring, and $m$ represent the sample number. Define a function set, denoted by $\mathcal{F}$, such that $\mathcal{F} : \mathcal{R}_q \to \mathcal{R}_q$ is an efficiently computable function set. Key-dependent pseudorandomness for NTRU encodings holds if for any* $\mathsf{PPT}$ *adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *satisfying*

$$\left| \Pr\left[ \begin{array}{c} \mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda, \mathbf{t}_b) \\ = b \end{array} : \begin{array}{c} b \leftarrow \{0,1\}, \mathbf{g} \leftarrow\$\ \chi^m \\ f \leftarrow\$\ \chi \textit{ with } f^{-1} \\ \mathbf{t}_0 \leftarrow \mathbf{g}\cdot f^{-1}, \mathbf{t}_1 \leftarrow\$\ \mathcal{R}_q^m \end{array} \right] - \frac{1}{2} \right| \le \mathsf{negl}(\lambda),$$

*where the oracle $\mathcal{O}$ takes as input a function $h \in \mathcal{F}$, and returns $(g_h + h(f)) \cdot f^{-1}$ with $g_h \leftarrow\$\ \chi$.*

Next, we demonstrate how to reduce the problem: if an adversary breaks the pseudorandomness given the public key, then we can construct an adversary that breaks Definition 5.

*Proof of pseudorandomness in Theorem 1.* We construct an adversary $\mathcal{B}$ against Definition 5, building on an adversary $\mathcal{A}$ against the pseudorandomness given the public key:

1. During initialization, $\mathcal{B}$ receives $\mathbf{t}_b$ from the challenger.

2. Using the oracle $\mathcal{O}$, $\mathcal{B}$ queries $\log_2 N$ times to construct $\{\mathbf{W}_i\}_{i\in[\log_2 N]}$. Specifically, for $j \in [t_{pk} := \lceil \log_{B_{pk}} q \rceil]$, it takes as input a function $h(x) = B_{pk}^j \cdot \tau_{2^i+1}(x)$ and receives the output $k_{i,j}$. Then, the $i^{\text{th}}$ key is combined as follows:

$$\mathbf{W}_i \leftarrow (k_{i,0}, k_{i,1}, \cdots, k_{i,t_{pk}-1}).$$

3. $\mathcal{B}$ simulates $(\mathbf{a}, \mathbf{t}_b, \{\mathbf{W}_i\}_{i\in[\log_2 N]})$ to $\mathcal{A}$.

4. Whatever $\mathcal{A}$ returns, $\mathcal{B}$ returns the same value.

Specifically, for each $i \in [\log_2 N]$, the above packing key satisfies the following:

$$\begin{aligned}\mathbf{W}_i \leftarrow & \ (k_{i,0}, k_{i,1}, \cdots, k_{i,t_{pk}-1}) \\ = & \ (\tfrac{g_0 + B_{pk}^0 \cdot \tau_{2^i+1}(f)}{f}, \cdots, \tfrac{g_{t_{pk}-1} + B_{pk}^{t_{pk}-1} \cdot \tau_{2^i+1}(f)}{f}) \\ = & \ \tfrac{(g_0, \cdots, g_{t_{pk}-1}) + \tau_{2^i+1}(f) \cdot (B_{pk}^0, \cdots, B_{pk}^{t_{pk}-1})}{f} \\ = & \ ((g_0, \cdots, g_{t_{pk}-1}) + \tau_{2^i+1}(f) \cdot \mathbf{g}_{B_{pk}}) \cdot f^{-1}.\end{aligned}$$

Therefore, $\mathcal{B}$ perfectly simulates the public key (i.e., the packing key) to $\mathcal{A}$, and then we can conclude that the advantage in Definition 5 is identical to the advantage in the pseudorandom game given the public key. We have completed the proof. □

## A.2 Proof of Theorem 2

*Proof of Theorem 2.* In this proof, we analyze all potential computational errors associated with answering the query, providing a comprehensive error analysis.

We divide the Response algorithm into four steps and analyze errors related to query recovery, database query multiplication, packing, and modulus switching.

For query recovery, we compute the coefficient expansion and recovery in steps 1-2 of Figure 4. As shown in Section 2.4, the coefficient expansion generates an error with the parameter $\sigma_{ce} \le \sqrt{\frac{1}{3}(2^r)^2 t_{ce} N B_{ce}^2 \sigma_\chi^2 + (2^r)^2 \sigma_\chi^2}$, where $r = \lceil \log_2 \ell \rceil$ and $t_{ce} = \lceil \log_{B_{ce}} q \rceil$. However, the noise in the original encoding is not amplified by the factor $(2^r)^2$ because it has already been multiplied by $((2^r)^{-1} \bmod q)$ in advance. Step 2 involves the external product, which introduces an additional error with a parameter of $\sigma_{gd} = \sqrt{\frac{1}{12} t_g N B_g^2 \sigma_\chi^2}$, where $t_g = \left\lceil \log_{B_g} q \right\rceil$. Due to the fact that $\|\mathsf{rp}\|_2^2 = 1$, the first error has a parameter of $\sigma_{1st} \le \sqrt{\frac{1}{3}(2^r)^2 t_{ce} N B_{ce}^2 \sigma_\chi^2 + \frac{1}{12} t_g N B_g^2 \sigma_\chi^2 + \sigma_\chi^2}$.

In multiplication, the query consists of $\ell$ NTRU encodings with an error parameter of $\sigma_{1st}$. For any $i \in [N\phi]$ and $j \in [\ell]$, the error $d_{i,j} \cdot \mathsf{qu}_j$ is defined as $d_{i,j} \cdot g_j$. Since each $d_{i,j}$ coefficient is uniformly distributed in $\mathbb{Z}_p$, the error distribution $d_{i,j} \cdot g_j$ has a parameter of $\sqrt{\frac{1}{12} N p^2 \sigma_{1st}^2}$. Therefore, the resulting error for each $c_j$ has a parameter of $\sigma_{2nd} = \sqrt{\frac{1}{12} N \ell p^2 \sigma_{1st}^2}$.

For the NTRU packing, an additional error is given by Theorem 1. After packing, the result is subject to an error with a parameter of $\sqrt{\frac{1}{36}(N^2-1) t_{pk} N B_{pk}^2 \sigma_\chi^2 + N^2 \sigma_{2nd}^2}$, where $t_{pk} = \left\lceil \log_{B_{pk}} q \right\rceil$. Similarly, factoring $N^{-1} \bmod q$ in advance can help avoid amplifying $N^2$, and the third error has a parameter of $\sigma_{3th} \le \sqrt{\frac{1}{36}(N^2-1) t_{pk} N B_{pk}^2 \sigma_\chi^2 + \sigma_{2nd}^2}$.

Finally, as described in Section 2.5, switching the modulus introduces an error $e_1$ satisfying $\|e_1\|_\infty \le \frac{1}{2}$ under Lemma 1. The variance of this error satisfies $\sigma_{4th}^2 \le \frac{1}{4} N \sigma_\chi^2$.

In summary, after four steps, the error in answers is sub-Gaussian, with a parameter of

$$\begin{aligned}\bar{\sigma}^2 := & \frac{\sigma_{3th}^2}{q_2^2} + \sigma_{4th}^2 \le \frac{N\sigma_\chi^2}{4} + \frac{(N^2-1) t_{pk} N B_{pk}^2 \sigma_\chi^2}{36 q_2^2} + \frac{\sigma_{2nd}^2}{q_2^2} \\ = & \frac{N\sigma_\chi^2}{4} + \frac{(N^2-1) t_{pk} N B_{pk}^2 \sigma_\chi^2}{36 q_2^2} + \frac{N \ell p^2 \sigma_{1st}^2}{12 q_2^2} \\ = & \frac{N\sigma_\chi^2}{4} + \frac{N\sigma_\chi^2}{4 q_2^2}\left(\frac{(N^2-1) t_{pk} B_{pk}^2}{9} + \frac{\ell p^2}{3}\left(1 + \frac{(2^r)^2 t_{ce} N B_{ce}^2}{3} + \frac{t_g N B_g^2}{12}\right)\right).\end{aligned}$$

Due to the sub-Gaussian distribution, the probability of a correctness error of size $\delta$ is constrained as follows:

$$\Pr[|X| > \tfrac{\Delta_1}{2}] < 2 \cdot \exp\left(-\pi \Delta_1^2 / 4\bar{\sigma}^2\right) \le \delta.$$

This inequality can be further converted into:

$$4\ln(\tfrac{\delta}{2})\bar{\sigma}^2 \geq -\pi\Delta_1^2 \Leftrightarrow \Delta_1 \geq 2\bar{\sigma}\cdot\sqrt{\frac{\ln(\frac{2}{\delta})}{\pi}} \Leftrightarrow$$

$$\begin{aligned}\lfloor q_1/p \rfloor \geq \sigma_\chi \cdot \sqrt{\frac{N\ln(\frac{2}{\delta})}{\pi}} \\ \cdot\sqrt{1+\frac{(\frac{(N^2-1)t_{pk}B_{pk}^2}{9}+\frac{\ell p^2}{3}(1+\frac{(2^r)^2 t_{ce}NB_{ce}^2}{3}+\frac{t_g NB_g^2}{12}))}{q_2^2}}.\end{aligned} \tag{2}$$

Thus, our NPIR scheme is $\delta$-correct as long as the above inequality condition is satisfied. □

## A.3 Proof of Theorem 3

*Proof of Theorem 3.* The proof of multi-query privacy for the NPIR scheme is based on the idea of proving single-query privacy and then extending it through hybrid games to multi-query privacy. The core of the proof lies in establishing the indistinguishability of two distributions:

$$\mathcal{D}_0 \coloneqq \{\mathsf{qu}_k \mid \mathsf{qu}_k \leftarrow \mathsf{Query}(\mathsf{qk}, \mathsf{ind}_{k,0})\}_{k\in[\mathcal{Q}]}$$

and

$$\mathcal{D}_1 \coloneqq \{\mathsf{qu}_k \mid \mathsf{qu}_k \leftarrow \mathsf{Query}(\mathsf{qk}, \mathsf{ind}_{k,1})\}_{k\in[\mathcal{Q}]},$$

given the expansion and packing keys. Suppose that the index set $\mathcal{I}$ consists of $\{(\mathsf{ind}_{k,0}, \mathsf{ind}_{k,1}) \coloneqq ((\mathsf{col}_{k,0}, \mathsf{term}_{k,0}), (\mathsf{col}_{k,1}, \mathsf{term}_{k,1})) \in ([\ell]\times[N])^2\}_{k\in[\mathcal{Q}]}$.

**Lemma 2** (Query privacy for a single query)**.** *Given a security parameter $\lambda$ and a single query with two different indexes* $(\mathsf{ind}_0, \mathsf{ind}_1)$*, Definitions 2 and 5 hold for the parameter set. Then, the* NPIR *scheme satisfies query privacy for a single query.*

We prove Lemma 2 by defining a series of hybrid games:

- $\mathsf{H}_0^b$: Same as the game in Definition 1. Specifically, the challenger first samples $(\mathsf{pp}, \mathsf{qk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and a bit $b \in \{0,1\}$, and sends $\mathsf{pp}$ to $\mathcal{A}$. When $\mathcal{A}$ queries with $(\mathsf{ind}_0, \mathsf{ind}_1)$, the challenger responds with $\mathsf{qu} \leftarrow \mathsf{Query}(\mathsf{qk}, \mathsf{ind}_b)$. Finally, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.
- $\mathsf{H}_1^b$: Same as $\mathsf{H}_0^b$ except that the challenger samples $\mathsf{qu}$.
- $\mathsf{H}_2^b$: Same as $\mathsf{H}_1^b$ except that the challenger samples $\mathsf{pk}, \mathsf{ck}$, which form the public parameters $\mathsf{pp}$.

Next, we show that the adjacent games are indistinguishable:

- $\mathsf{H}_0^b$ and $\mathsf{H}_1^b$ are indistinguishable under Definition 5 with respect to two oracles with functions $h_0(x) = B_{pk}^j \cdot \tau_{2^i+1}(x)$ and $h_1(x) = B_{ce}^j \cdot \tau_{2^i+1}(x)$. We observe that the expansion and packing keys are both from the $\mathsf{AutoSetup}$ algorithm, which can be simulated by oracles with $h_0(x)$ and $h_1(x)$. Furthermore, the query can be viewed as a set of NTRU encodings ($t_g + 1$ samples). Therefore, we conclude that the queries in the two games are indistinguishable under Definition 5.
- $\mathsf{H}_1^b$ and $\mathsf{H}_2^b$ are indistinguishable under Definition 2. Both keys are generated from the $\mathsf{AutoSetup}$ algorithm in NTRU encoding format. Through a hybrid reduction to Definition 2, we conclude that the output keys are indistinguishable from uniform. In this case, all information interacted in $\mathsf{H}_2^b$ is random and uncorrelated with $b$. For $\mathcal{A}$, the probability of winning the game is equal to the probability of a random guess, which is $\frac{1}{2}$.

Thus, Definitions 2 and 5 hold for the parameter set, and then Lemma 2 holds. We then use hybrid games to extend Lemma 2 to Theorem 3:

- $\mathsf{G}_0$: Same as the game with the query distribution $\mathcal{D}_0$.
- For $j \in [1 : \mathcal{Q}]$, $\mathsf{G}_j$: Same as $\mathsf{G}_{j-1}$ except that the $(j-1)^{\text{th}}$ query is derived from $\mathcal{D}_1$. When $j = \mathcal{Q}$, $\mathsf{G}_j$ is the game with the query distribution $\mathcal{D}_1$.

The difference between two neighboring games, labeled $\mathsf{G}_j$ and $\mathsf{G}_{j+1}$, lies in the distribution of the $j^{\text{th}}$ query. According to Lemma 2, we have that, for $j \in [\mathcal{Q}]$,

$$\mathsf{G}_j \overset{\textit{Lemma } 2}{\Longleftrightarrow} \mathsf{G}_{j+1}.$$

Suppose that the advantage in Lemma 2 is a negligible probability $\varepsilon$ and the advantage in each $\mathsf{G}_j$ is $p_j$. Using the difference lemma, we can show that, for any $j \in [\mathcal{Q}]$,

$$|p_j - p_{j-1}| \leq \varepsilon.$$

Furthermore, we have the inequality that

$$|p_{\mathcal{Q}} - p_0| \leq |p_{\mathcal{Q}} - p_{\mathcal{Q}-1}| + \cdots + |p_1 - p_0| \leq \mathcal{Q} \cdot \varepsilon,$$

which is a negligible probability.

In summary, we have proven that the Npir scheme has query privacy for any set of indexes of $\mathcal{Q}$ queries. □