VIA: COMMUNICATION-EFFICIENT SINGLE-SERVER PRIVATE INFORMATION RETRIEVAL

CHENYANG LIU*†, XUKUN WANG*†, ZHIFANG ZHANG*
EMAIL: {LIUCHENYANG23, WANGXUKUN, ZFZ}@AMSS.AC.CN

Abstract. Private Information Retrieval (PIR) is a crucial component in many privacy-preserving systems, with Offline/Online PIR attracting significant attention. Recent works have focused on eliminating offline communication overhead. However, existing constructions incur high online communication costs as a trade-off. To address this, we propose VIA, a single-server PIR scheme that eliminates offline communication while achieving $O_\lambda(\log N)$ online communication complexity. Experimental evaluations demonstrate that for a 32 GB database, VIA requires only 690 KB of online communication—a 3.7× reduction compared to state-of-the-art schemes without offline communication—while attaining a throughput of 3.11 GB/s. Furthermore, we introduce VIA-C, a variant of VIA that allows offline communication. Compared to previous communication-efficient schemes, VIA-C achieves a 24.5× reduction in online communication, requiring only 2.1 KB for a 32 GB database (with 14.8 MB offline communication). Moreover, VIA-C can naturally extend to VIA-B that supports batch queries. Compared to previous communication-efficient batch PIR schemes, VIA-B achieves a 3.5× reduction in query size and a 127× reduction in response size for a 1 GB database of 1-byte records. The designs of our schemes rely on a novel DMux-CMux structure and LWE-to-RLWE conversion techniques.

1. Introduction

Private Information Retrieval (PIR), first introduced by Chor, Goldreich, Kushilevitz, and Sudan [1], enables a client to retrieve the i-th entry of a database from a server while keeping i private. PIR has found extensive applications in various domains, such as anonymous communication [2, 3], contact discovery [4], private certificate transparency auditing[5], secure web browsing [6], and privacy-preserving media streaming [7].

Communication overhead is a crucial performance metric for PIR schemes. This work focuses on optimizing the communication cost of existing PIR schemes under various settings. Classical PIR schemes [1, 8, 9, 10, 11] have no offline phase. Although [10, 9, 11] achieve logarithmic communication complexity, they often suffer from prohibitively large constant factors or high computational costs, rendering them impractical. Offline/Online PIR schemes [10, 12, 13, 14, 15, 5, 16, 17, 18, 19, 3, 20] optimize online computation time and communication efficiency by introducing an additional offline phase. SimplePIR[5] represents the state-of-the-art in high-throughput offline/online PIR. However, it requires expensive offline communication and computation.

Challenges of hints. The offline/online framework, while amortizing costs over multiple queries, suffers from several drawbacks: primarily, client-side storage requirements for database-dependent hints become prohibitive [5], especially when querying multiple databases or in storage-constrained settings, forcing premature hint eviction before cost amortization is achieved. Moreover, maintaining correctness necessitates recomputing all hints upon database updates, imposing significant communication

*State Key Laboratory of Mathematical Sciences, AMSS, Chinese Academy of Sciences.
*School of Mathematical Sciences, University of Chinese Academy of Sciences.
†These authors contributed equally as first authors.
This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFA0712300 and in part by the Chinese Academy of Sciences Project for Young Scientists in Basic Research under Grant YSBR-008. (Corresponding author: Xukun Wang.)

overhead that undermines amortization benefits, particularly for highly dynamic databases. Additionally, certain schemes [12, 13, 14] require clients to upload client-specific state, forcing servers to store large public keys per client, which not only incurs significant storage overhead but also demands additional infrastructure for efficient client state lookup and retrieval, while potentially exposing the system to active attacks if client-specific public keys are reused [5].

PIR without offline communication. Recent preprocessing PIR schemes aim to eliminate offline communication [21, 22, 20]. These efforts face a fundamental trade-off: while removing offline communication, they incur significant online overhead. Tiptoe [21] and HintlessPIR [22] essentially leverage a form of "bootstrapping" [23] to homomorphically compress the hint from SimplePIR [5], avoiding client hint downloads but introducing computational and communication cost. Additionally, YPIR [20] uses a key-switching approach to compact DoublePIR[5] responses via polynomial rings, substantially reducing communication costs.

The schemes above leverage RLWE assumptions and LWE-to-RLWE conversions to eliminate offline communication, but at the expense of high online communication. HintlessPIR [22] and YPIR [20] achieve high throughput by fixing the random part of ciphertexts, requiring distinct client secret keys per query and transmission of non-precomputable ciphertext conversion parameters. This results in $O_\lambda(\sqrt{N})$ online communication costs [22, 20]. It appears that lattice-based PIR without offline communication inherently incurs high online communication costs, raising a challenging question: Can we design a high-throughput, communication-efficient PIR protocol without offline communication?

The VIA protocol. This work introduces the VIA protocol, a lattice-based single-server PIR protocol without offline communication, achieving $\widetilde{O}_\lambda(1)$ online communication complexity. Similar to most PIR schemes that eliminate offline communication [21, 22, 20], the VIA scheme guarantees that the clients do not need to store any database-dependent hint, and the server does not need to store any client-specific information. The design of VIA primarily employs the DMux-CMux framework and ring-switching technique, eliminating offline communication while achieving logarithmic online communication complexity and high online efficiency. For a 1 GB database, VIA achieves higher throughput than the state-of-the-art silent preprocessing scheme YPIR while reducing online communication by 1.8×. For a 32 GB database, VIA's throughput is 1.5× lower than YPIR but reduces online communication by 3.7×.

PIR with offline communication. XPIR [19], SealPIR [12], OnionPIR [13], Spiral [14], and Respire[15] reduce the online communication cost by allowing offline communication. SealPIR and OnionPIR introduce query compression algorithms to reduce query size, while Spiral proposes new compression and ciphertext conversion techniques to further reduce the communication cost. Respire improves Spiral's query compression technique and introduces a novel response compression algorithm using ring-switching, further lowering communication for small-record databases.

The VIA-C protocol. This work introduces VIA-C, a variant of VIA with offline communication, constituting a communication-efficient lattice-based single-server PIR scheme. VIA-C relies on the MLWE assumption. Similarly to VIA, VIA-C employs the same DMux-CMux structure to reduce the encrypted information from $O(\sqrt{N})$ to $O(\log N)$. Additionally, VIA-C uses a novel LWE-to-RLWE conversion-based query compression technique, shrinking the query vector to only $O(\log N)$ elements in $\mathbb{Z}_q$ (where q is the ciphertext modulus). For instance, for a 32 GB database, VIA-C's query size is only 0.659 KB, whereas the state-of-the-art communication-efficient PIR Respire requires 57.77 KB (87.7× larger). Moreover, VIA-C adopts blinded extraction to compress the response into a single polynomial instead of a full RLWE ciphertext, further reducing the response size. VIA-C also leverages Respire's ring-switching technique for response compression. Notably, for a 32 GB database, VIA-C reduces the total online communication by 24.5× compared to Respire while achieving 2.26× higher throughput. Compared to high-throughput protocols like SimplePIR, VIA-C is 7.6× slower but reduces the communication cost by 690×. Furthermore, SimplePIR requires expensive preprocessing time (3376.67 seconds for a 32 GB database), whereas VIA-C only needs 67.539 seconds.

1.1. **Main Techniques.** The substantial communication overhead in lattice-based PIR stems from the large lattice parameters required to achieve correctness and security guarantees. Our goal is to minimize the online communication cost while preserving correctness and security. As shown in Figure 1, the VIA protocol eliminates offline communication and significantly reduces online communication overhead by replacing coefficient expansion in [24, 14, 15] with the DMux technique. When offline communication is permitted, the VIA-C scheme that adopts a novel LWE-to-RLWE conversion algorithm, substantially reduces online communication compared to existing schemes [12, 5, 24, 14, 15].

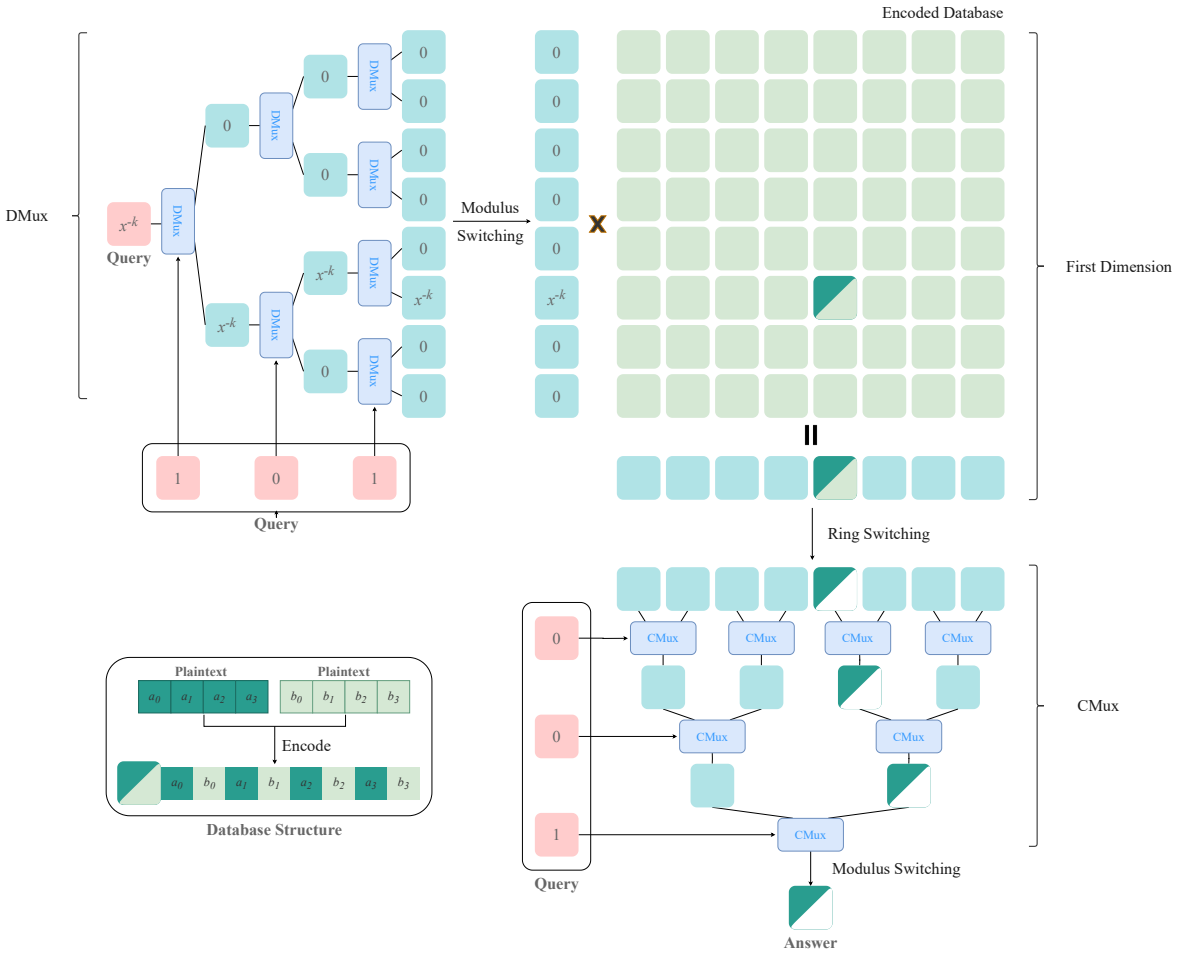

FIGURE 1. A toy example of VIA for a database of size $N = 128$ and numbers of rows and columns are both 8. The record that the client wants to retrieve is indexed by $(6, 5)$. We refer to Section 3 for formal descriptions and full details.

DMux-CMux structure. In schemes such as Spiral[14] and Respire[15], the database is represented as a matrix, and the required encrypted unit vectors are generated via coefficient expansion. This approach incurs prohibitively high communication costs, requiring an offline phase to partially offset them. In contrast, our scheme employs DMux operations as a substitute for coefficient expansion, which requires only logarithmically many ciphertexts (relative to the number of database rows) to generate an encrypted one-hot vector. This design eliminates offline communication. After plaintext-ciphertext matrix-vector multiplication yields a specific row, the CMux technique is then applied to homomorphically select the desired record. Notably, this DMux-CMux architecture resembles a symmetric triangular configuration, which inspired the name VIA for our scheme.

LWE-to-RLWE conversion. Efficient encryption of constants under RLWE is crucial for lattice-based PIR [13, 14, 15]. Although prior work proposes LWE-to-RLWE conversion to avoid direct RLWE encryption [25, 26, 27], these schemes become impractical for PIR applications due to their large public parameters. Methods with compact parameters [28] support packing, and amortize $O(1)$ key switching operations, but incur cubic noise growth in the polynomial degree n. To overcome this, we introduce a new LWE-to-RLWE conversion that achieves logarithmic-sized public parameters and logarithmic key switching operations, adding only $O(n \log n)$ noise variance. Our LWE-to-RLWE conversion is functionally equivalent to HERMES[29] in the case where HERMES is provided with a single LWE ciphertext and a set of zero encryptions. However, unlike HERMES's multi-ciphertext packing, our method focuses on single-ciphertext settings, avoiding unnecessary computational overhead. Secondly, the key-switching methods are different. Thirdly, HERMES typically uses few MLWE midpoints (only one in experiments) to maintain computational efficiency, while our scheme employs $\log n$ midpoints, reducing both computation and public parameter size. VIA-C implements this novel LWE-to-RLWE conversion algorithm that achieves enhanced query compression—reducing queries to $O(\log N)$ elements in $\mathbb{Z}_q$. For example, for a 32 GB database, the query size is only 0.659 KB.

MLWEs-to-RLWE conversion. We generalize the LWE-to-RLWE conversion to MLWEs-to-RLWE conversion and employ it as a foundational building block for our homomorphic repacking technique. Our homomorphic repacking algorithm introduces only logarithmic noise variance, making it suitable for large-scale ciphertext repacking scenarios. VIA-B utilizes this novel homomorphic repacking algorithm to substantially reduce response sizes during large-batch queries on databases with tiny records (1-byte). For a 1 GB database with 256 batch queries, the response size is approximately 1.81 KB. Additionally, through appropriate secret key selection, the public parameters for homomorphic repacking align with those for query compression; consequently, employing homomorphic repacking in VIA-B incurs no additional public parameters.

Starting point: the Respire protocol. The VIA protocol is based on the Respire protocol [15] (the lattice-based PIR protocol with the best communication for small records). Briefly, Respire represents the database as an $I \times J$ matrix. The query consists of four components: an encrypted length-I one-hot vector for selecting the target row; the encrypted selection bits for selecting the desired column; the rotation bits for shifting the result; and the ring-switching key for extracting and compressing the final answer. Respire achieves significant communication reduction by compressing the first three query components through coefficient expansion while reusing the ring-switching key.

VIA and VIA-C protocols. The VIA protocol replaces coefficient expansion with DMux for generating the encrypted length-I one-hot vector, thereby eliminating offline communication. Simultaneously, by employing fresh ciphertexts that relax lattice parameter constraints, VIA allows a larger plaintext modulus. VIA-C further reduces online communication by compressing the queries in VIA using a novel LWE-to-RLWE conversion-based query compression algorithm. For a 32 GB database, online communication is reduced by 329× compared to VIA and by 28.5× compared to Respire.

1.2. **Our Contributions.** In summary, our contributions are:

- a scheme eliminating offline communication with only $O(\log N)$ online communication cost while maintaining high throughput. (Section 3)
- a novel LWE-to-RLWE conversion method with lower noise growth. (Section 4.1)
- a communication-efficient scheme employing innovative query compression and response optimization techniques. (Section 4.4)
- an evaluation of these schemes, which achieve high communication efficiency, and anonymously open-sourced at `https://anonymous.4open.science/r/VIA-8888/`. (Section 5)

2. Preliminaries

For $N \in \mathbb{N}$, let $[N] = \{0, 1, \ldots, N-1\}$. For $i \in [N]$, denote by $\mathbf{i}$ the binary representation of i and $\mathrm{rev}(\mathbf{i})$ is the reversed version. Every tuple $\mathbf{t} = (t_0, t_1, \ldots, t_{N-1})$ is zero-indexed and represented by a bold letter, where t_i represents the i-th element of $\mathbf{t}$ for each $i \in [N]$.

For a probability distribution $\mathcal{P}$, we write $x \leftarrow \mathcal{P}$ to denote that x is randomly sampled from $\mathcal{P}$. For a set S, we write $x \leftarrow S$ to denote that x is sampled uniformly at random from S.

2.1. **Private Information Retrieval.** A single-server PIR scheme with preprocessing for databases over plaintext space $\mathcal{D}$ of length N consists of five phases:

- $\mathrm{Setup}(1^\lambda) \to (\mathrm{pp}, \mathrm{qk})$: Given a security parameter λ, output the public parameters pp and a query key qk.
- $\mathrm{SetupDB}(1^\lambda, \mathbf{X}) \to \mathbf{db}$: Given a security parameter λ and a database $\mathbf{X} \in \mathcal{D}^N$, output an encoded database $\mathbf{db}$.
- $\mathrm{Query}(\mathrm{idx}, \mathrm{qk}) \to (\mathrm{qu}, \mathrm{st})$: Given an index $\mathrm{idx} \in [N]$ and the query key qk, output a query qu and a secret state st.
- $\mathrm{Answer}(\mathrm{pp}, \mathbf{db}, \mathrm{qu}) \to \mathrm{ans}$: Given the parameters pp, the encoded database $\mathbf{db}$ and the query qu, output an answer ans.
- $\mathrm{Recover}(\mathrm{qk}, \mathrm{st}, \mathrm{ans}) \to d$: Given the query key qk, the secret state st and the answer ans, output a record $d \in \mathcal{D}$.

Correctness. A single-server PIR scheme with preprocessing is correct with error δ if for any database $\mathbf{X} \in \mathcal{D}^N$ and for any index $i \in [N]$,

$$\Pr\left[\begin{array}{rl} & (\mathrm{pp}, \mathrm{qk}) \leftarrow \mathrm{Setup}(1^\lambda) \\ & \mathbf{db} \leftarrow \mathrm{SetupDB}(1^\lambda, \mathbf{X}) \\ d \neq X_i : & (\mathrm{qu}, \mathrm{st}) \leftarrow \mathrm{Query}(i, \mathrm{qk}) \\ & \mathrm{ans} \leftarrow \mathrm{Answer}(\mathrm{pp}, \mathbf{db}, \mathrm{qu}) \\ & d \leftarrow \mathrm{Recover}(\mathrm{qk}, \mathrm{st}, \mathrm{ans}) \end{array}\right] \leq \delta.$$

Security. A single-server PIR scheme with preprocessing is secure if for all $(i, j) \in [N]^2$, the conditional distributions $(\mathrm{Query}(i, \mathrm{qk})|\,\mathrm{pp})$ and $(\mathrm{Query}(j, \mathrm{qk})|\,\mathrm{pp})$ are computationally indistinguishable.

2.2. **Lattice-Based Homomorphic Encryption.** The Learning With Errors (LWE) problem was first proposed by Regev in 2005 [30]. Its ring variant, known as the Ring-LWE problem, was introduced by Lyubashevsky, Peikert, and Regev in 2010 [31]. The efficient single-server PIR protocols [2, 32, 12, 33, 19, 13, 34] are constructed using lattice-based homomorphic encryption schemes [23, 35, 36, 37, 38, 39]. These lattice-based FHE systems fundamentally rely on the LWE assumption and the RLWE assumption.

RLWE. RLWE, as a ring version of the LWE problem, is parameterized by (n, q, χ_S, χ_E), where n is the polynomial degree that is a power of 2, q is a modulus, and χ_S is the distribution of the key over $\mathcal{R}_{n,q} = \mathbb{Z}_q[X]/(X^n + 1)$ and χ_E is the distribution of error over $\mathcal{R}_{n,q}$. The RLWE assumption asserts that for a polynomial $A \leftarrow \mathcal{R}_{n,q}$, a secret key $S \leftarrow \chi_S$, an error polynomial $E \leftarrow \chi_E$, and a random polynomial $R \leftarrow \mathcal{R}_{n,q}$, the following distributions are computationally indistinguishable:

$$\{(A, A \cdot S + E)\} \approx_c \{(A, R)\}.$$

RLWE-based homomorphic encryption. The RLWE-based homomorphic encryption scheme consists of the following algorithms:

- $\mathrm{ParamGen}(1^\lambda)$: Given the security parameter, output a parameter tuple $(n, p, q, \chi_S, \chi_E)$.
- $\mathrm{KGen}(\chi_S)$: Sample and output $S \leftarrow \chi_S$.
- $\mathrm{Ecd}(m)$: Given a plaintext polynomial $m \in \mathcal{R}_{n,p}$, output an encoded polynomial $M = \Delta m \in \mathcal{R}_{n,q}$, where $\Delta = \lceil q/p \rceil$.
- $\mathrm{Enc}_S(M)$: For $M \in \mathcal{R}_{n,q}$, sample $A \leftarrow \mathcal{R}_{n,q}$ and $E \leftarrow \chi_E$, then output $c = (A, B) = (A, AS + E + M)$.
- $\mathrm{Dec}_S(c)$: Output $\lfloor (B - AS)/\Delta \rceil$.

This encryption scheme supports the following basic homomorphic operations:

- **Addition.** Given ciphertexts $c_1 \leftarrow \mathrm{Enc}_S(M_1)$ and $c_2 \leftarrow \mathrm{Enc}_S(M_2)$, output $c_1 + c_2 \in \mathrm{Enc}_S(M_1 + M_2)$.
- **Plaintext-Ciphertext Multiplication.** Given a plaintext polynomial $M \in \mathcal{R}_{n,q}$ with small coefficients, and $c \leftarrow \mathrm{Enc}_S(M_1)$, output $M \cdot c \in \mathrm{Enc}_S(M \cdot M_1)$.

We denote the distribution of encryption of M by $\mathrm{RLWE}_S(M)$, and for a tuple $\mathbf{M} \in \mathcal{R}_{n,q}^m$, we denote $\mathrm{RLWE}_S(\mathbf{M}) = \prod_{i\in[m]} \mathrm{RLWE}_S(M_i)$.

Approximate gadget decomposition. Similar to several recent PIR protocols, our construction also relies on the Gentry, Sahai, and Waters (GSW) encryption scheme to achieve homomorphic multiplication. We adopt the approximate gadget decomposition proposed in [40]: for gadget parameters (ℓ, B), decompose M as

$$M \approx M_1 \frac{q}{B^1} + M_2 \frac{q}{B^2} + \cdots + M_\ell \frac{q}{B^\ell},$$

with $M_1, M_2, \ldots, M_\ell \in \mathcal{R}_{n,B}$. We denote $\mathrm{Decomp}(M)$ as the tuple $(M_1, M_2, \ldots, M_\ell)$.

RLev. The RLev ciphertext was first introduced in [41] to designate an intermediate type of ciphertext between RLWE and RGSW ciphertexts. An RLev ciphertext $\hat{c}$ under the secret key S is a tuple of RLWE ciphertexts encrypting the same message $M \in \mathcal{R}_{n,q}$ with gadget parameters:

$$\hat{c} \in \prod_{i=1}^{\ell} \mathrm{RLWE}_S(qM/B^i) = \mathrm{RLev}_S(M).$$

RGSW. An RGSW ciphertext C encrypting $M \in \mathcal{R}_{n,q}$ under the secret key S is a pair of RLev ciphertexts:

$$C \in \mathrm{RLev}_S(-SM) \times \mathrm{RLev}_S(M) = \mathrm{RGSW}_S(M).$$

For $\mathbf{M} \in \mathcal{R}_{n,q}^k$ of polynomials, we denote $\mathrm{RGSW}_S(\mathbf{M}) = \prod_{i=0}^{k-1} \mathrm{RGSW}_S(M_i)$. We remark that the gadget parameters in $\mathrm{RLev}_S(-SM)$ and $\mathrm{RLev}_S(M)$ could be different.

Gadget product. The gadget product takes as input a polynomial $M_1 \in \mathcal{R}_{n,q}$ and an RLev ciphertext $\hat{c} \leftarrow \mathrm{RLev}_S(M_2)$, and outputs the gadget product $M_1 \odot \hat{c} \in \mathrm{RLWE}_S(M_1 M_2)$, which is denoted as:

$$\begin{aligned} \odot : M_1 \times \mathrm{RLev}_S(M_2) &\to \mathrm{RLWE}_S(M_1 M_2) \\ M_1 \odot \hat{c} &\mapsto \langle \mathrm{Decomp}(M_1), \hat{c} \rangle. \end{aligned}$$

External product. The external product takes as input an RGSW ciphertext $C = (\hat{c}_1, \hat{c}_2) \leftarrow \mathrm{RGSW}_S(M_1)$, and an RLWE sample $c = (A, B) \leftarrow \mathrm{RLWE}_S(M_2)$, and then outputs an RLWE sample of message $M_1 M_2$, which is denoted as

$$\begin{aligned} \boxdot : \mathrm{RGSW}_S(M_1) \times \mathrm{RLWE}_S(M_2) &\to \mathrm{RLWE}_S(M_1 M_2) \\ C \boxdot c &\mapsto A \odot \hat{c}_1 + B \odot \hat{c}_2. \end{aligned}$$

RLWE key-switching. Given an RLWE ciphertext $c = (A, B) \leftarrow \mathrm{RLWE}_S(M)$ under secret key S and to re-encrypt M under a new key S', the key-switching key ksk is needed, which is an RLev ciphertext $\mathrm{ksk} \in \mathrm{RLev}_{S'}(S)$. The key-switching algorithm outputs $c' \leftarrow \mathrm{KeySwitch}(c, \mathrm{ksk})$ where

$$c' = (0, B) - A \odot \mathrm{ksk} \in \mathrm{RLWE}_{S'}(M).$$

Sample extraction. For an RLWE ciphertext $c = (A, B) \leftarrow \mathrm{RLWE}_S(M)$ where $M = \sum_{i=0}^{n-1} m_i X^i$, $A = \sum_{i=0}^{n-1} a_i X^i$, $B = \sum_{i=0}^{n-1} b_i X^i$ and $S = \sum_{i=0}^{n-1} s_i X^i$, the sample extraction algorithm outputs an LWE ciphertext $\mathrm{Extr}(c) = (a_0, -a_{n-1}, \ldots, -a_1, b_0)$ of message m_0 under key $(s_0, s_1, \ldots, s_{n-1})$.

Modulus switching. Given an RLWE ciphertext $c = (A, B)$ over $\mathcal{R}_{n,q_1}$ and smaller moduli q_2, q_3, output a pair $\mathrm{ModSwitch}_{q_2,q_3}(c) = (\lfloor q_2 A/q_1 \rceil, \lfloor q_3 B/q_1 \rceil)$ over $\mathcal{R}_{n,q_2}$.

3. The VIA Protocol

In this section we present the VIA protocol. We first introduce the building blocks of VIA.

Ring switching. Similar to the Respire[15], we also adopt the ring switching algorithm from [42, 37, 35]. Specifically, we employ a larger ring $\mathcal{R}_{n_1,q}$ for RLWE encoding and utilize its subring $\mathcal{R}_{n_2,q}$ to pack $d = n_1/n_2$ database records. We first introduce the following functions:

- The embedding function: for $j \in [d]$,

$$\iota_j^{n_2 \to n_1} : \mathcal{R}_{n_2} \to \mathcal{R}_{n_1}$$
$$\sum_{i=0}^{n_2-1} f_i x^i \mapsto \sum_{i=0}^{n_2-1} f_i x^{d \cdot i + j},$$

  and define $\iota^{n_2 \to n_1} : \mathcal{R}_{n_2,q}^d \to \mathcal{R}_{n_1,q}$ by $\iota^{n_2 \to n_1}(\mathbf{F}) = \sum_{j=0}^{d-1} \iota_j^{n_2 \to n_1}(F_j)$.
- The projection function: for $j \in [d]$,

$$\pi_j^{n_1 \to n_2} : \mathcal{R}_{n_1} \to \mathcal{R}_{n_2}$$
$$\sum_{i=0}^{n_1-1} f_i x^i \mapsto \sum_{i=0}^{n_2-1} f_{d \cdot i + j} x^i,$$

  and define $\pi^{n_1 \to n_2} : \mathcal{R}_{n_1,q} \to \mathcal{R}_{n_2,q}^d$ by $\pi^{n_1 \to n_2}(F) = (\pi_0^{n_1 \to n_2}(F), \ldots, \pi_{d-1}^{n_1 \to n_2}(F))$.

Abstractly, the ring switching algorithm is a homomorphic projection. Given secret keys $S_1 \in \mathcal{R}_{n_1,q}$ and $S_2 \in \mathcal{R}_{n_2,q}$, it takes as input an RLWE ciphertext $c \leftarrow \mathrm{RLWE}_{S_1}(M)$ and a ring-switching key rsk, outputs a ciphertext $\mathrm{RingSwitch}(c, \mathrm{rsk}) \in \mathrm{RLWE}_{S_2}(\pi_0^{n_1 \to n_2}(M))$ over ring $\mathcal{R}_{n_2,q}$.

CMux. VIA employs CMux gates for homomorphic selection. The Mux function acts as an if condition: it takes two inputs, a select bit b and a pair (a_0, a_1) of options. Depending on the value of the select bit, it outputs one of the two options. Formally, the Mux function is defined as $\mathrm{Mux}(b, (a_0, a_1)) = a_b$, where $b \in \{0, 1\}$ is the select bit. This function can be algebraically implemented as $\mathrm{Mux}(b, (a_0, a_1)) = a_0 + b(a_1 - a_0)$.

The homomorphic Mux, denoted as CMux [40], can be constructed via the external product.

Let $c_i \leftarrow \mathrm{RLWE}_S(M_i)$ for $i = 0, 1$, and let encrypted select bit $C \in \mathrm{RGSW}_S(b)$ with $b \in \{0, 1\}$, then

$$\mathrm{CMux}(C, c_0, c_1) = c_0 + C \boxdot (c_1 - c_0) \in \mathrm{RLWE}_S(M_b).$$

The 1-out-of-2 CMux function can be naturally extended to an 1-out-of-2^m CMux function. To select the x-th element, the selection bits should be the reversed binary representation of x, as described in algorithm 1.

Algorithm 1 1-out-of-2^m CMux

Input: Encrypted select bits $\mathbf{C}_{\mathrm{sel}} \leftarrow \mathrm{RGSW}_S(\mathbf{b})$ with $\mathbf{b} \in \{0, 1\}^m$, an RLWE data $\mathbf{c} \leftarrow \mathrm{RLWE}_S(\mathbf{db})$ with $\mathbf{db} \in \mathcal{R}_{n,p}^{2^m}$.

Output: A ciphertext of $\mathrm{RLWE}_S(\mathrm{db}_{\mathrm{rev}(\mathbf{b})})$.

```
res^0 ← c.
for i ← 0 to m − 1 do
    for j ← 0 to 2^{m−i} − 1 do
        res_j^{i+1} = CMux(C_{sel,i}, res_{2j}^i, res_{2j+1}^i)
    end for
end for
return res^m
```

DMux. A demultiplexer (DMux) is a digital circuit that directs a single input signal to one of multiple output lines, based on control bits. The 1-to-2 homomorphic DMux can be implemented as follows: for an encrypted control bit $C \in \mathrm{RGSW}_S(b)$ with $b \in \{0, 1\}$, and $c \leftarrow \mathrm{RLWE}_S(M)$,

$$\mathrm{DMux}(C, c) = (c - C \boxdot c, C \boxdot c).$$

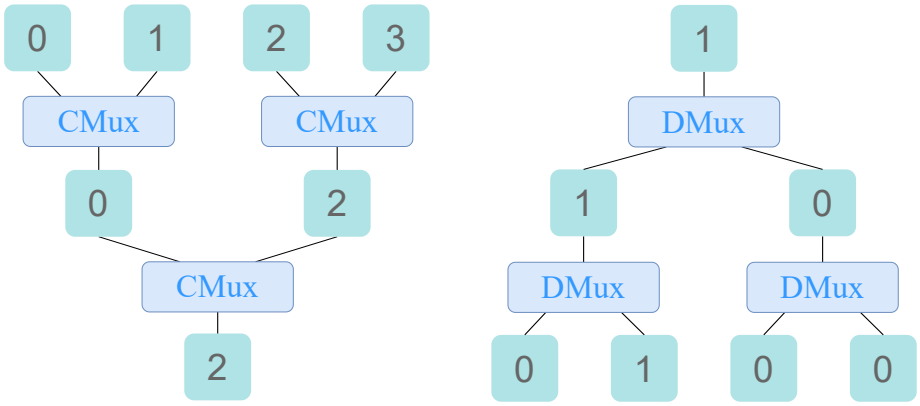

FIGURE 2. CMux-DMux Structure with select bits 01 and control bits 01.

Similarly, the 1-to-2 DMux function can be naturally extended to a 1-to-2^m DMux function, as visually shown in Figure 2. A similar algorithm is discussed in [43, 44]. Consider a binary tree structure with depth $m = \log N$. During the initialization phase, all nodes are set to 0 except for the root node, which is initialized to c. The target index y is represented in its binary representation, with each bit individually encrypted. These encrypted bits serve as control bits, one for each level of the tree. After performing $2^m - 1$ external product operations, the value of the root node is propagated to the leaf node corresponding to the target index, as formally described in Algorithm 2.

Algorithm 2 1-to-2^m DMux

Input: Encrypted control bits $\mathbf{C}_{\text{ctrl}} \leftarrow \text{RGSW}_S(\mathbf{b})$ with $\mathbf{b} \in \{0,1\}^m$, a root value $c \leftarrow \text{RLWE}_S(M)$.
Output: 2^m RLWE ciphertexts of message M in the $\mathbf{b}$-th position and of message 0 in others.

$\text{res}^0 \leftarrow c$.
for $i \leftarrow 0$ **to** $m-1$ **do**
  for $j \leftarrow 0$ **to** $2^i - 1$ **do**
    $(\text{res}_{2j}^{i+1}, \text{res}_{2j+1}^{i+1}) = \text{DMux}(C_{\text{ctrl},i}, \text{res}_j^i)$
  end for
end for
return res^m

3.1. **The VIA Protocol.** From a conceptual perspective, VIA employs an architecture similar to [24, 14, 15]. Crucially, VIA substitutes coefficient expansion techniques with DMux for generating the encrypted one-hot vector, introducing only logarithmic noise variance in the one-hot vector length. In contrast, conventional approaches employing ciphertext compression techniques [24, 14, 15] exhibit at least quadratic noise growth. Furthermore, existing schemes require servers to maintain client-specific public parameters for communication efficiency, whereas VIA eliminates public parameters entirely.

Notably, we implement modulus switching before first-dimensional folding. This optimization significantly reduces computational overhead for first-dimensional folding while incurring negligible additional error.

Similar to [15, 14], VIA employs CMux gates for homomorphic selection. However, existing schemes incur substantial noise growth through compressed control-bit ciphertexts. VIA addresses this limitation by using fresh control bits, ring switching to a smaller ring before CMux. This approach simultaneously reduces both ciphertext size and computational complexity.

Database structure. Let λ be a security parameter. Let q_1, q_2, q_3, q_4, p be moduli with $q_1 > q_2 > q_3 > q_4 > p$ and n_1, n_2 be polynomial degrees with $n_2 \mid n_1$.

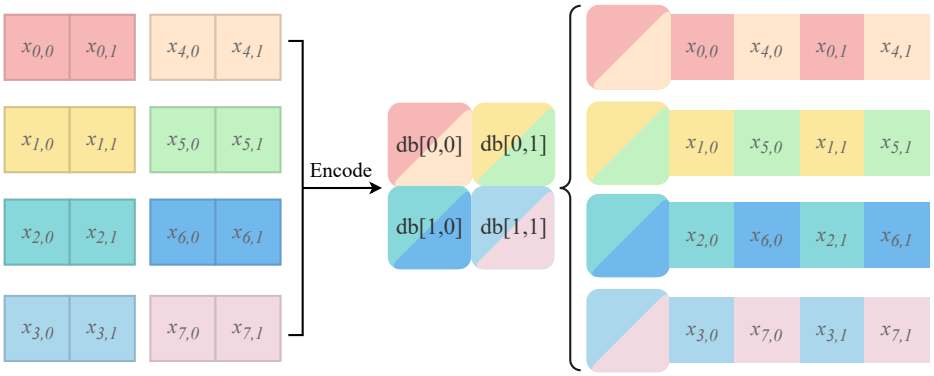

FIGURE 3. Database Structure

In VIA, the ciphertexts over R_{n_1,q_1}, R_{n_1,q_2}, R_{n_2,q_3}, and R_{n_2,q_4} will be utilized. The initial database is over ring $\mathcal{R}_{n_2,p}$. For a database $\mathbf{X} \in \mathcal{R}_{n_2,p}^N$, it is structured as an $I \times J$ matrix over $\mathcal{R}_{n_1,p}$ with $Nn_2 = IJn_1$. We index the database record by a pair $(i,j) \in [I] \times [J]$. Specifically, the (i,j)-record

$$\mathrm{db}[i,j] = \iota^{n_2 \to n_1}((\mathrm{db}_{kIJ+iJ+j})_{k \in [n_1/n_2]}).$$

The construction of VIA is given in Fig. 4.

Correctness. In Appendix B, we describe how we choose the scheme parameters concretely (to achieve a target error rate of 2^{-40}).

Security. In the VIA protocol (Figure 4), the query consists of a ring switching key from S_1 to S_2, a ciphertext of rotation and a tuple of RGSW ciphertext of control bits under secret key S_1 and a tuple of RGSW ciphertext of select bits under secret key S_2. Assuming the underlying homomorphic encryption system is secure along with a "circular security" assumption, we deduce that VIA is secure.

3.2. **Further Optimizations.**

Reduce encrypted control bits size. To generate the encrypted one-hot vector, the server needs a rotation ciphertext c_{rot} and $\log I$ RGSW ciphertexts of control bits. Noting that the size of RGSW ciphertexts is significantly larger than that of regular RLWE ciphertexts, the client can send the results of the t-th level DMux, which is 2^t RLWE ciphertexts, instead of the ciphertext of rotation and the ciphertexts of control bits of the first t layers. This optimization effectively reduces communication overhead, computational complexity, and noise accumulation throughout the process.

Reduce encrypted select bits size. Analogously, observing that the answer ans is significantly smaller than RGSW ciphertexts over R_{n_2,q_3}, the client only sends the select bits of the initial levels of CMux, and the server computes only the initial levels of CMux and sends the result as the answer to the client. This optimization reduces communication overhead, computational complexity, and noise accumulation throughout the process. Furthermore, this strategy enables VIA to retrieve more information per query, thereby expanding its capacity to larger records.

Blinded extraction. In the query phase, while the rotation ciphertext $c_{\mathrm{rot}} \leftarrow \mathrm{Enc}(x^{-\gamma})$ conventionally assumes $\gamma \in [n_1/n_2]$, the client is in fact permitted to select any $\gamma \in [n_1]$. When specifically retrieving the t-th coefficient of a database record, the client may send $c_{\mathrm{rot}} \leftarrow \mathrm{Enc}(x^{-t})$. The server then executes the answer algorithm identically, computes ans, and performs sample extraction $\mathrm{ans}_{\mathrm{ext}} \leftarrow \mathrm{Extr}(\mathrm{ans})$, which is returned as the final response. Furthermore, the approach naturally generalizes to multi-sample extraction scenarios, enabling VIA to flexibly support databases with variable-sized records.

- SetupDB$(1^\lambda, \mathbf{X} \in \mathcal{R}_{n_2,p}^N) \to \mathbf{db}$: Given a security parameter λ and the initial database $\mathbf{X}$, the setup algorithm computes $\mathrm{db}[i,j] = \iota^{n_2 \to n_1}((X_{kIJ+iJ+j})_{k \in [n_1/n_2]}) \in \mathcal{R}_{n_1,p}$ for all $i \in [I], j \in [J]$ and outputs the entire encoded database $\mathbf{db} \in \mathcal{R}_{n_1,p}^{I \times J}$.
- Query$(\mathrm{idx}) \to (\mathrm{qu}, \mathrm{st})$: Given the index $\mathrm{idx} \in [N]$, sample secret keys $S_1 \leftarrow \chi_{S,1}$ and $S_2 \leftarrow \chi_{S,2}$, and generate a ring switching key rsk from S_1 to S_2. The query algorithm computes $(\alpha, \beta, \gamma) \in [I] \times [J] \times [n_1/n_2]$ such that $\mathrm{idx} = \gamma IJ + \alpha J + \beta$. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ be the binary representations of α and β respectively. Then the query algorithm encrypts $c_{\mathrm{rot}} \leftarrow \mathrm{RLWE}_{S_1}(X^{-\gamma})$, $\mathbf{C}_{\mathrm{ctrl}} \leftarrow \mathrm{RGSW}_{S_1}^{(\ell_1,B_1),(\ell_1',B_1')}(\boldsymbol{\alpha})$ and $\mathbf{C}_{\mathrm{sel}} \leftarrow \mathrm{RGSW}_{S_2}(\mathrm{rev}(\boldsymbol{\beta}))$, and it outputs $\mathrm{qu} = (c_{\mathrm{rot}}, \mathbf{C}_{\mathrm{ctrl}}, \mathrm{rsk}, \mathbf{C}_{\mathrm{sel}})$ and query key $\mathrm{st} = S_2$.
- Answer$(\mathbf{db}, \mathrm{qu}) \to \mathrm{ans}$: Given the encoded database $\mathbf{db} \in \mathcal{R}_{n_1,p}^{I \times J}$ and the query qu, the answer algorithm proceeds as follows:
  1. **Homomorphic demultiplexer:** Compute $(c_0^{(0)}, c_1^{(0)}, \ldots, c_{I-1}^{(0)}) \leftarrow \mathrm{DMux}(\mathbf{C}_{\mathrm{ctrl}}, c_{\mathrm{rot}})$.
  2. **Modulus switching:** For each $i \in [I]$, compute $c_i^{(1)} \leftarrow \mathrm{ModSwitch}_{q_2,q_2}(c_i^{(0)})$.
  3. **First dimension:** For each $j \in [J]$, compute $c_j^{(2)} = \sum_{i \in [I]} c_i^{(1)} \mathrm{db}[i,j]$.
  4. **Ring switching:** For each $j \in [J]$, compute $c_j^{(3)} \leftarrow \mathrm{RingSwitch}(\mathrm{rsk}, c_j^{(2)})$.
  5. **Homomorphic multiplexer:** Compute $\mathrm{ans}' \leftarrow \mathrm{CMux}(\mathbf{C}_{\mathrm{sel}}, (c_j^{(3)})_{j \in J})$.
  6. **Modulus switching:** Return $\mathrm{ans} \leftarrow \mathrm{ModSwitch}_{q_3,q_4}(\mathrm{ans}')$.
- Recover$(\mathrm{st}, \mathrm{ans}) \to d$: Given the query key st and the answer ans, output $d = \mathrm{Dec}_{\mathrm{st}}(\mathrm{ans})$.

FIGURE 4. The VIA Protocol

4. The VIA-C Protocol

In this section, we introduce a variant of VIA with offline communication, termed VIA-C. At a high level, VIA-C compresses the queries in VIA into LWE ciphertexts, utilizing the LWE-to-RLWE and RLWE-to-RGSW conversions. We first present the fundamental building blocks for constructing VIA-C, followed by the detailed VIA-C protocol in Subsection 4.4.

Before presenting our LWE-to-RLWE conversion, we first introduce the MLWE assumption and the MLWE-based homomorphic encryption system.

MLWE assumption. The MLWE assumption, first introduced in [37], is a generalization of LWE assumption and RLWE assumption, which is parameterized by $(m, n, q, \chi_S, \chi_E)$, where m is the MLWE rank, n is the polynomial degree that is a power of 2, q is a modulus, and χ_S is the distribution of the key over $\mathcal{R}_{n,q}$ and χ_E is the distribution of error over $\mathcal{R}_{n,q}$. The MLWE assumption asserts that for polynomials $\mathbf{A} \leftarrow \mathcal{R}_{n,q}^m$, secret keys $\mathbf{S} \leftarrow \chi_S^m$, an error polynomial $E \leftarrow \chi_E$, and random polynomial $R \leftarrow \mathcal{R}_{n,q}$, the following distributions are computationally indistinguishable:

$$\{(\mathbf{A}, \langle \mathbf{A}, \mathbf{S} \rangle + E)\} \approx_c \{(\mathbf{A}, R)\}.$$

The MLWE problem reduces to the LWE assumption when $n = 1$ and to the RLWE assumption when $m = 1$. Furthermore, the $(m, n, q, \chi_S, \chi_E)$-MLWE problem is at least as hard as the (mn, q, χ_S, χ_E)-RLWE problem.

MLWE encryption. The (m, n)-MLWE encryption of a polynomial $M \in \mathcal{R}_{n,q}$ under key $\mathbf{S} \in \mathcal{R}_{n,q}^m$ is given by the tuple $(\mathbf{A}, B) \in \mathcal{R}_{n,q}^{m+1}$, where $\mathbf{A} \leftarrow \mathcal{R}_{n,q}^m$ and $B = \langle \mathbf{A}, \mathbf{S} \rangle + E + M \in \mathcal{R}_{n,q}$. The random variable corresponding to this encryption of polynomial M is denoted by $\mathrm{MLWE}_{\mathbf{S}}^{m,n}(M)$, the superscript m, n may be omitted when the parameters (m, n) are non-critical.

4.1. LWE-to-RLWE Conversion.

In lattice-based PIR, the necessity to encrypt constants under RLWE arises frequently. However, encrypting constants directly under RLWE is highly inefficient. To address this, various forms of LWE-to-RLWE conversion techniques have been explored in [25, 26, 28, 27]. Unfortunately, the large public parameters of [25, 26, 27] render them impractical for PIR applications.

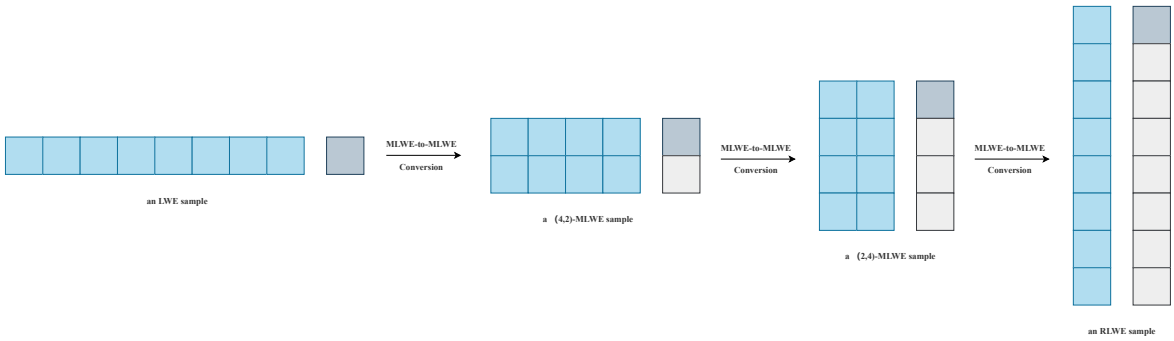

FIGURE 5. An Example of LWE-to-RLWE Conversion when $n = 8$.

In contrast, the LWE-to-RLWE conversion approach from [28] features compact public parameters and supports ciphertext packing, achieving an amortized $O(1)$ key switching operations. Consequently, [24, 14, 15, 20] adopt either the LWE-to-RLWE conversion in [28] or its coefficient extraction variant. However, a drawback of this method is its cubic noise variance growth in polynomial degree n. Employing coefficient extraction within VIA would lead to substantial impracticalities: either prohibitively difficult parameter selection or support only for very small plaintext modulus.

To overcome these limitations, we propose a novel LWE-to-RLWE conversion technique that achieves logarithmic key-switching keys, logarithmic key switching operations, and introduces only $O(n \log n)$ noise variance.

RLWE-to-MLWE extraction. Sample extraction in Section 2 can be naturally extended to RLWE-to-MLWE extraction. Given a ciphertext $c = (A, B) \leftarrow \mathrm{RLWE}_S(M)$ over $\mathcal{R}_{n,q}$ and an integer $d \mid n$, the RLWE-to-MLWE extraction algorithm outputs an $(n/d, d)$-MLWE ciphertext

$$\mathrm{Extr}_d(c) = (\pi_0^{n\to d}(A), \pi_{n/d-1}^{n\to d}(A)X, \ldots, \pi_1^{n\to d}(A)X, \pi_0^{n\to d}(B))$$

under key $\pi^{n\to d}(S)$ of message $\pi_0^{n\to d}(M)$.

MLWE-to-RLWE insertion. Given an MLWE ciphertext $c = (\mathbf{A}, B) \leftarrow \mathrm{MLWE}_{\mathbf{S}}^{m,n}(M)$, the MLWE-to-RLWE insertion outputs an RLWE ciphertext

$$\mathrm{Insert}(c) = (\iota^{n\to nm}(A_0, A_{m-1}X^{-1}, \cdots, A_1X^{-1}), \iota_0^{n\to nm}(B))$$

under the key $\iota^{n\to nm}(\mathbf{S})$, and the 0-th projection of the corresponding message is M.

MLWE key switching. MLWE key switching is formulated through the integration of MLWE-to-RLWE insertion, RLWE key switching, and RLWE-to-MLWE extraction. Given MLWE keys $\mathbf{S}, \mathbf{S}' \in \mathcal{R}_{n,q}^m$, an RLWE key-switching key ksk from $\iota^{n\to nm}(\mathbf{S})$ to $\iota^{n\to nm}(\mathbf{S}')$ and an MLWE ciphertext $c \leftarrow \mathrm{MLWE}_{\mathbf{S}}^{m,n}(M)$, the MLWE key-switching algorithm outputs

$$\mathrm{KeySwitch}(c, \mathrm{ksk}) = \mathrm{Extr}_n(\mathrm{KeySwitch}(\mathrm{Insert}(c), \mathrm{ksk})) \in \mathrm{MLWE}_{\mathbf{S}'}^{m,n}(M).$$

MLWE embedding. For an integer d that is a power of 2, the MLWE embedding takes as input an (m, n)-MLWE ciphertext $c = (\mathbf{A}, B) \leftarrow \mathrm{MLWE}_{\mathbf{S}}^{m,n}(M) \in \mathcal{R}_{n,q}^{m+1}$, outputs an (m, nd)-MLWE ciphertext

$$\mathrm{Embed}_d(c) = (\iota_0^{n\to nd}(A_0), \cdots, \iota_0^{n\to nd}(A_{m-1}), \iota_0^{n\to nd}(B)) \in \mathcal{R}_{nd,q}^{m+1}$$

of $\iota_0(M)$ under key $(\iota_0^{n\to nd}(S_0), \cdots, \iota_0^{n\to nd}(S_{m-1}))$.

MLWE-to-MLWE conversion. We first present the MLWE-to-MLWE conversion, which serves as the fundamental building block for our LWE-to-RLWE conversion. This algorithm transforms an (md, n)-MLWE ciphertext into an (m, nd)-MLWE ciphertext.

For an integer d that is a power of 2, and an (md, n)-MLWE ciphertext $c = (\mathbf{A}, B) \leftarrow \mathrm{MLWE}_{\mathbf{S}}(M) \in \mathcal{R}_{n,q}^{md+1}$, then for $i \in [d]$, $c_{(i)} = (A_{mi}, A_{mi+1}, \ldots, A_{mi+m-1}, 0) \in \mathcal{R}_{n,q}^{m+1}$ can be viewed as an (m, n)-MLWE ciphertext under key $\mathbf{S}_{(i)} = (S_{mi}, S_{mi+1}, \ldots, S_{mi+m-1})$, applying MLWE embedding we get an MLWE ciphertext $c^{(i)} = \mathrm{Embed}_d(c_{(i)}) \in \mathcal{R}_{nd,q}^{m+1}$ under key $\mathbf{S}^{(i)} = (\iota_0^{n\to nd}(S_{mi}), \cdots, \iota_0^{n\to nd}(S_{mi+m-1}))$. For any key $\mathbf{S}' \in \mathcal{R}_{nd,p}^{m}$, let $\mathrm{ksk}^{(i)}$ be a key-switching key from $\mathbf{S}^{(i)}$ to $\mathbf{S}'$, then

$$\mathrm{Conv}_d(c, \{\mathrm{ksk}^{(i)}\}) = (\mathbf{0}, B') + \sum_{i=0}^{d-1} \mathrm{KeySwitch}(c^{(i)}, \mathrm{ksk}^{(i)})$$

is an (m, nd)-MLWE ciphertext of $\iota_0(M)$ under key $\mathbf{S}'$, where $B' = \iota_0^{n\to nd}(B)$. Furthermore, we state the following lemma, the proof of which is presented in Appendix A.

Lemma 4.1. *Under the notations above, let θ_c be the variance of error in c, θ_{ks} be the variance of error introduced by key-switching and θ' be the variance of error in* $\mathrm{Conv}_d(c, \{\mathrm{ksk}^{(i)}\})$. *Then $\theta' \le \theta_c + d\theta_{\mathrm{ks}}$.*

LWE-to-RLWE conversion. Let n be a power of 2, and let $c \leftarrow \mathrm{LWE}_{\mathbf{s}}(\mu)$ be a rank n LWE ciphertext, where $\mu \in \mathbb{Z}_q$. Viewing c as an $(n, 1)$-MLWE ciphertext, we iteratively apply the MLWE-to-MLWE conversion: first obtaining an $(n/2, 2)$-MLWE ciphertext for $\iota_0(\mu) = \mu$, then an $(n/4, 4)$-MLWE ciphertext for $\iota_0(\mu) = \mu$, and so forth. This process ultimately yields a $(1, n)$-MLWE ciphertext $\mathrm{LWEtoRLWE}(c)$ for μ, equivalent to an RLWE ciphertext, as visually shown in Figure 5.

The LWE-to-RLWE algorithm requires $2\log n$ key-switching keys and performs $2\log n$ key-switching operations in total. Consequently, we establish the following lemma, the proof of which appears in Appendix A.

Lemma 4.2. *Let θ_c be the variance of error in c, θ_{ks} be the variance of error introduced by key-switching and θ' be the variance of error in* $\mathrm{LWEtoRLWE}(c)$. *Then $\theta' \le \theta_c + 2\theta_{\mathrm{ks}} \log n$.*

As key switching introduces $O(n)$ noise variance, Lemma 4.2 demonstrates that the LWE-to-RLWE conversion algorithm above incurs $O(n \log n)$ noise variance. In contrast, the algorithm presented in [28] introduces noise variance $\theta' \le n^2\theta_c + \frac{n^2-1}{3}\theta_{\mathrm{ks}}$, corresponding to $O(n^3)$ noise variance.

We denote the key-switching keys used in the LWE-to-RLWE conversion by $\hat{c}_{\mathrm{toRLWE}}$, and denote this procedure as $\mathrm{LWEtoRLWE}(c, \hat{c}_{\mathrm{toRLWE}})$.

4.2. **Query Compression.**

CRot. To accomplish ring switching while keeping γ private, homomorphic rotation of ciphertexts is required. VIA implements this by incorporating $c_{\mathrm{rot}} \leftarrow \mathrm{RLWE}_S(X^{-\gamma})$ into the query through plaintext-ciphertext multiplication. To reduce communication overhead in VIA-C, direct transmission of RLWE ciphertexts is avoided. Instead, the homomorphic rotation methodology employed in Respire [15] is adopted, denoted as Controlled Rotation (CRot). The CRot algorithm takes as input a tuple $\mathbf{C}_{\mathrm{rot}} \leftarrow \mathrm{RGSW}_S(\gamma_i)$ of RGSW ciphertexts with $\gamma_i \in \{0, 1\}$, and an RLWE ciphertext $c \leftarrow \mathrm{RLWE}_S(M)$, outputs an RLWE ciphertext $\mathrm{CRot}(\mathbf{C}_{\mathrm{rot}}, c) \in \mathrm{RLWE}_S(MX^{-\sum_i \gamma_i 2^i})$.

RLWE-to-RGSW conversion. The RLWE-to-RGSW algorithm introduced in Spiral [14] is realized through external multiplication. This technique enables the client to transmit exclusively the RLev ciphertext of M, thereby obtaining its RGSW ciphertext form. VIA-C similarly employs this methodology, albeit utilizing the gadget product implementation from [45], which eliminates the need for external multiplication while introducing a relatively small error. The RLWE-to-RGSW conversion algorithm takes as input a ciphertext $c = (A, B) \leftarrow \mathrm{RLWE}_S(M)$ and an RLev ciphertext $\hat{c}_{\mathrm{toRGSW}} \leftarrow \mathrm{RLev}_S(S^2)$, outputs an RLWE ciphertext $\mathrm{RLWEtoRGSW}(c, \hat{c}_{\mathrm{toRGSW}}) = (B, 0) + A \odot \hat{c}_{\mathrm{toRGSW}} \in \mathrm{RLWE}_S(-SM)$.

Query compression. Upon implementing CRot, queries in VIA-C consist of three components: the encrypted rotation bits, the encrypted control bits, and the encrypted selection bits. They are all RGSW ciphertexts of values in $\{0,1\}$. Through application of the RLWE-to-RGSW conversion, communication is optimized by transmitting exclusively the RLev ciphertexts. Note that for any $b \in \{0,1\}$, $\mathrm{RLev}_S(b) = \prod_{i=1}^{\ell} \mathrm{RLWE}(bq/B^i)$ comprises ℓ RLWE ciphertexts of constants. Subsequent to LWE-to-RLWE conversion, VIA-C queries are compactly represented as $\ell \log(IJn_1/n_2) = \ell \log N$ LWE ciphertexts. Given that each LWE ciphertext contains precisely one non-random element, pseudorandomness techniques permit further reduction. Consequently, VIA-C queries necessitate merely $\ell \log N$ elements in $\mathbb{Z}_{q_1}$. The detailed construction of the query compression algorithm is illustrated in Figure 6.

- $\mathrm{QueryCompSetup}(1^\lambda, \mathbf{s}, S) \to \mathrm{pp}_{\mathrm{qck}}$: On input a security parameter and keys $\mathbf{s} \in \mathbb{Z}_{q_1}^n$ and $S \in \mathcal{R}_{n,q_1}$, the setup algorithm generates the LWE-to-RLWE conversion key $\hat{c}_{\mathrm{toRLWE}}$ and the RLWE-to-RGSW key $\hat{c}_{\mathrm{toRGSW}}$), then outputs $\mathrm{pp}_{\mathrm{qck}} = (\hat{c}_{\mathrm{toRLWE}}, \hat{c}_{\mathrm{toRGSW}})$.
- $\mathrm{QueryComp}(\mathbf{s}, \mathbf{b}_{\mathrm{ctrl}}, \mathbf{b}_{\mathrm{sel}}, \mathbf{b}_{\mathrm{rot}}) \to \mathrm{qu}$: On input a key $\mathbf{s} \in \mathbb{Z}_{q_1}^n$ and inputs $\mathbf{b}_{\mathrm{ctrl}} \in \{0,1\}^{\log I}$, $\mathbf{b}_{\mathrm{sel}} \in \{0,1\}^{\log J}$, $\mathbf{b}_{\mathrm{rot}} \in \{0,1\}^{\log\lceil n_1/n_2 \rceil}$, for all $1 \le i \le \ell$, the query compression algorithm encrypts $q_1\mathbf{b}_{\mathrm{ctrl}}/B^i$, $q_1\mathbf{b}_{\mathrm{sel}}/B^i$, $q_1\mathbf{b}_{\mathrm{rot}}/B^i$ as LWE ciphertexts and outputs the LWE ciphertexts as query qu.
- $\mathrm{QueryDecomp}(\mathrm{pp}_{\mathrm{qck}}, \mathrm{qu}) \to \mathbf{C}_{\mathrm{ctrl}}, \mathbf{C}_{\mathrm{sel}}, \mathbf{C}_{\mathrm{rot}}$: On input the public parameters $\mathrm{pp}_{\mathrm{qck}}$ and a query qu, for every LWE ciphertext c in qu the query decompression algorithm computes $c' \leftarrow \mathrm{LWEtoRLWE}(c, \hat{c}_{\mathrm{toRLWE}})$ and $c'' \leftarrow \mathrm{RLWEtoRGSW}(c', \hat{c}_{\mathrm{toRGSW}})$, then combines those RLWE ciphertexts as RGSW ciphertexts $\mathbf{C}_{\mathrm{ctrl}}$, $\mathbf{C}'_{\mathrm{sel}}$, $\mathbf{C}'_{\mathrm{rot}}$ of length $\log I$, $\log J$, $\log(n_1/n_2)$, respectively. Then, computes and outputs $\mathbf{C}_{\mathrm{sel}} \leftarrow \mathrm{ModSwitch}_{q_2,q_2}(\mathbf{C}'_{\mathrm{sel}})$ and $\mathbf{C}_{\mathrm{rot}} \leftarrow \mathrm{ModSwitch}_{q_2,q_2}(\mathbf{C}'_{\mathrm{rot}})$.

FIGURE 6. Query Compression Algorithms

4.3. **Response Compression.** In VIA, ring switching is employed following first-dimensional folding to reduce communication overhead and computational cost, leveraging freshly encrypted selection bits. VIA-C circumvents this by directly applying CMux after the first-dimensional folding, followed by CRot and ring switching to the CMux output. This approach simultaneously avoids the need for any additional LWE-to-RLWE conversion parameters and prevents error accumulation inherent to CRot operations. Consequently, VIA-C utilizes a response compression algorithm analogous to Respire's implementation, with the detailed procedure depicted in Figure 7.

- $\mathrm{RespCompSetup}(1^\lambda, S_1, S_2) \to \mathrm{pp}_{\mathrm{rck}}$: On input a security parameter and keys $S_1 \in \mathcal{R}_{n_1,q_3}$ and $S_2 \in \mathcal{R}_{n_2,q_3}$, the setup algorithm generates and outputs the ring-switching key $\mathrm{pp}_{\mathrm{rck}}$ from S_1 to S_2.
- $\mathrm{RespComp}(\mathrm{pp}_{\mathrm{rck}}, c) \to \mathrm{ans}$: On input the public parameter $\mathrm{pp}_{\mathrm{rck}}$ and a ciphertext $c \leftarrow \mathrm{RLWE}_{S_1}(M) \in \mathcal{R}^2_{n_1,q_2}$, the response compression algorithm computes $c' \leftarrow \mathrm{ModSwitch}_{q_3,q_3}(c)$, and computes $(A,B) \leftarrow \mathrm{RingSwitch}(c', \mathrm{pp}_{\mathrm{rck}})$, then output $\mathrm{ans} \leftarrow (A, \lfloor q_4 B/q_3 \rceil)$.
- $\mathrm{RespCompRecover}(S_2, \mathrm{ans}) \to d$: On input the key S_2 and a answer $\mathrm{ans} = (A', B')$, the response compression recover algorithm computes and outputs $\lfloor p(B' - \lfloor q_4 A' S_2/q_3 \rceil)/q_4 \rceil$.

FIGURE 7. Response Compression Algorithms

4.4. **The VIA-C Protocol.** We now describe the VIA-C protocol. At a high level, VIA-C is a variant of VIA with query compression and response compression techniques to reduce communication. The database structure and parameters of VIA-C protocol are the same as VIA. We give the VIA-C construction in Figure 8.

- Setup$(1^\lambda) \to (\mathrm{pp}, \mathrm{qk})$: Given the security parameter λ, the setup algorithm proceeds as follows:
  - Sample secret keys $\mathbf{s} \leftarrow \chi_s^{n_1}$, $S_1 \leftarrow \chi_{S,1}$ and $S_2 \leftarrow \chi_{S,2}$.
  - Sample parameters $\mathrm{pp}_{\mathrm{qck}} \leftarrow \mathrm{QueryCompSetup}(1^\lambda, S_1, S_2)$ and $\mathrm{pp}_{\mathrm{rck}} \leftarrow \mathrm{RespCompSetup}(1^\lambda, S_1, S_2)$.

  Output the query key $\mathrm{qk} = (\mathbf{s}, S_1, S_2)$ and the public parameters $\mathrm{pp} = (\mathrm{pp}_{\mathrm{qck}}, \mathrm{pp}_{\mathrm{rck}})$.
- SetupDB$(1^\lambda, \mathbf{X} \in \mathcal{R}_{n_2,p}^N) \to \mathbf{db}$: Given the security parameter λ and the initial database X of length N, the setup algorithm computes $\mathrm{db}[i,j] = \iota^{n_2 \to n_1}((X_{kIJ+iJ+j})_{k \in [n_1/n_2]}) \in \mathcal{R}_{n_1,p}$ for all $i \in [I], j \in [J]$ and outputs the entire encoded database $\mathbf{db} \in \mathcal{R}_{n_1,p}^{I \times J}$.
- Query$(\mathrm{qk}, \mathrm{idx}) \to (\mathrm{qu}, \mathrm{st})$: Given the query key $\mathrm{qk} = (\mathbf{s}, S_1, S_2)$ and the index $\mathrm{idx} \in [N]$, the query algorithm sets $\mathrm{st} = \perp$ and computes $(\alpha, \beta, \gamma) \in [I] \times [J] \times [n_1/n_2]$ such that $\mathrm{idx} = \gamma IJ + \alpha J + \beta$, then outputs $\mathrm{qu} \leftarrow \mathrm{QueryComp}(\mathbf{s}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ where $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are the binary representations of α, β and γ, respectively.
- Answer$(\mathrm{pp}, \mathbf{db}, \mathrm{qu}) \to \mathrm{ans}$: Given the parameter $\mathrm{pp} = (\mathrm{pp}_{\mathrm{qck}}, \mathrm{pp}_{\mathrm{rck}})$, an encoded database $\mathbf{db} \in \mathcal{R}_{n_1,p}^{I \times J}$, and a query qu, the answer algorithm proceeds as follows:
  1. **Query decompression:** Compute the decompressed query $\mathbf{C}_{\mathrm{ctrl}}, \mathbf{C}_{\mathrm{sel}}, \mathbf{C}_{\mathrm{rot}} \leftarrow \mathrm{QueryDecomp}(\mathrm{pp}_{\mathrm{qck}}, \mathrm{qu})$.
  2. **Homomorphic demultiplexer:** Compute $(c_0^{(0)}, c_1^{(0)}, \ldots, c_{I-1}^{(0)}) \leftarrow \mathrm{DMux}(\mathbf{C}_{\mathrm{ctrl}}, \lfloor q_1/p \rfloor)$.
  3. **Modulus switching:** For each $i \in [I]$, compute $c_i^{(1)} \leftarrow \mathrm{ModSwitch}_{q_2,q_2}(c_i^{(0)})$.
  4. **First dimension:** For each $j \in [J]$, compute $c_j^{(2)} = \sum_{i \in [I]} c_i^{(1)} \, \mathrm{db}[i,j]$.
  5. **Homomorphic multiplexer:** Compute $c^{(3)} \leftarrow \mathrm{CMux}(\mathbf{C}_{\mathrm{sel}}, (c_j^{(2)})_{j \in J})$.
  6. **Homomorphic rotation:** Compute $c^{(4)} \leftarrow \mathrm{CRot}(\mathbf{C}_{\mathrm{rot}}, c^{(3)})$.
  7. **Response compression:** Compute $\mathrm{ans} \leftarrow \mathrm{RespComp}(\mathrm{pp}_{\mathrm{rck}}, c^{(4)})$.
- Recover$(\mathrm{qk}, \mathrm{st}, \mathrm{ans}) \to d$: Given the query key $\mathrm{qk} = (\mathbf{s}, S_1, S_2)$, the secret state $\mathrm{st} = \perp$ and the answer ans, output $d \leftarrow \mathrm{RespCompRecover}(S_2, \mathrm{ans})$.

FIGURE 8. The VIA-C Protocol

4.5. **VIA-B: Extending VIA-C to the Batch Setting.** Analogous to Respire, the VIA-C protocol is extensible to batch query settings, termed VIA-B. The Respire packing methodology hinges upon two fundamental elements: homomorphic repacking and vectorization. Crucially, Respire's repacking algorithm utilizes homomorphic automorphisms, exhibiting a quadratic increase in noise variance corresponding to the number of ciphertexts to be packed. This limitation renders it impractical for small record sizes. We introduce a novel homomorphic repacking algorithm that achieves logarithmic noise variance scaling with respect to the number of ciphertexts to be packed.

Database configuration. In VIA-B, we model each database record as an element of $\mathcal{R}_{n_3,p}$ where $n_3 \leq n_2$ is a power-of-two. In VIA-B, each packed database element contains n_1/n_3 individual records.

4.6. **Homomorphic Repacking.** By generalizing the LWE-to-RLWE conversion, we can obtain the homomorphic repacking algorithm, or alternatively termed MLWEs-to-RLWE conversion.

MLWEs embedding. For an integer d that is a power of 2, the MLWE embedding takes as input d MLWE ciphertexts $c_i = (\mathbf{A}^i, B^i) \leftarrow \mathrm{MLWE}_{\mathbf{S}}^{m,n}(M^i) \in \mathcal{R}_{n,q}^{m+1}$, outputs an (m, nd)-MLWE ciphertext $\mathrm{Embed}_d(\mathbf{c}) = (\iota^{n \to nd}(A_0^0, \cdots, A_0^{d-1}), \cdots, \iota^{n \to nd}(A_{m-1}^0, \cdots, A_{m-1}^{d-1}), \iota^{n \to nd}(B^0, \cdots, B^{d-1}))$ of $\iota^{n \to nd}(M^0, \cdots, M^{d-1})$.

MLWEs-to-MLWE conversion. Extension of the MLWE-to-MLWE conversion algorithm—incorporating d MLWE ciphertexts as inputs while substituting the MLWE embedding with MLWEs embeddings—enables derivation of the MLWEs-to-MLWE conversion methodology. This algorithm transforms d many (md, n)-MLWE ciphertexts of $\mathbf{M} \in \mathcal{R}_{n,q}^d$ into an (m, nd)-MLWE ciphertext of message $\iota^{n \to nd}(\mathbf{M}) \in \mathcal{R}_{nd,q}$.

- Setup$(1^\lambda) \to (\mathrm{pp}, \mathrm{qk})$: Given the security parameter λ, the setup algorithm proceeds as follows:
  - Sample secret keys $\mathbf{s} \leftarrow \chi_s^{n_1}$, $S_1 \leftarrow \chi_{S,1}$ and $S_2 \leftarrow \chi_{S,2}$.
  - Sample parameters $\mathrm{pp}_{\mathrm{qck}} \leftarrow \mathrm{QueryCompSetup}(1^\lambda, S_1, S_2)$ and $\mathrm{pp}_{\mathrm{rck}} \leftarrow \mathrm{RespCompSetup}(1^\lambda, S_1, S_2)$.

  Output the query key $\mathrm{qk} = (\mathbf{s}, S_1, S_2)$ and the public parameters $\mathrm{pp} = (\mathrm{pp}_{\mathrm{qck}}, \mathrm{pp}_{\mathrm{rck}})$.
- SetupDB$(1^\lambda, \mathbf{X} \in \mathcal{R}_{n_3,p}^N) \to \mathbf{db}$: Given the security parameter λ and a database X of length N, the setup algorithm computes $\mathrm{db}[i,j] = \iota^{n_3 \to n_1}((X_{kIJ+iJ+j})_{k \in [n_1/n_3]}) \in \mathcal{R}_{n_1,p}$ for all $i \in [I], j \in [J]$ and outputs the entire encoded database $\mathbf{db} \in \mathcal{R}_{n_1,p}^{I \times J}$.
- Query$(\mathrm{qk}, \{\mathrm{idx}\}_{t \in [T]}) \to (\mathrm{qu}, \mathrm{st})$: Given the query key $\mathrm{qk} = (\mathbf{s}, S_1, S_2)$ and the index $\mathrm{idx}_t \in [N]^T$. For each $t \in [T]$, the query algorithm sets $\mathrm{st} = \perp$ and computes $(\alpha_t, \beta_t, \gamma_t) \in [I] \times [J] \times [n_1/n_3]$ such that $\mathrm{idx}_t = \gamma_t IJ + \alpha_t J + \beta_t$, then outputs $\mathrm{qu}_t \leftarrow \mathrm{QueryComp}(\mathbf{s}, \boldsymbol{\alpha}_t, \boldsymbol{\beta}_t, \boldsymbol{\gamma}_t)$ where $\boldsymbol{\alpha}_t, \boldsymbol{\beta}_t$ and $\boldsymbol{\gamma}_t$ are the binary representations of α_t, β_t and γ_t respectively.
- Answer$(\mathrm{pp}, \mathbf{db}, \mathbf{qu}) \to \mathrm{ans}$: Given the parameter $\mathrm{pp} = (\mathrm{pp}_{\mathrm{qck}}, \mathrm{pp}_{\mathrm{rck}})$, an encoded database $\mathbf{db} \in \mathcal{R}_{n_1,p}^{I \times J}$, and a query $\mathbf{qu}$, the answer algorithm proceeds as follows: for each $t \in [T]$,
  1. **Query decompression:** Compute the decompressed query $\mathbf{C}_{\mathrm{ctrl},t}, \mathbf{C}_{\mathrm{sel},t}, \mathbf{C}_{\mathrm{rot},t} \leftarrow \mathrm{QueryDecomp}(\mathrm{pp}_{\mathrm{qck}}, \mathrm{qu}_t)$.
  2. **Homomorphic demultiplexer:** Compute $(c_0^{(0),t}, c_1^{(0),t}, \ldots, c_{I-1}^{(0),t}) \leftarrow \mathrm{DMux}(\mathbf{C}_{\mathrm{ctrl},t}, \lfloor q/p \rfloor)$.
  3. **Modulus switching:** For each $i \in [I]$, compute $c_i^{(1),t} \leftarrow \mathrm{ModSwitch}_{q_2,q_2}(c_i^{(0),t})$.
  4. **First dimension:** For each $j \in [J]$, compute $c_j^{(2),t} = \sum_{i \in [I]} c_i^{(1),t} \, \mathrm{db}[i,j]$.
  5. **Homomorphic multiplexer:** Compute $c^{(3),t} \leftarrow \mathrm{CMux}(\mathbf{C}_{\mathrm{sel},t}, (c_j^{(2),t})_{j \in J})$.
  6. **Homomorphic rotation:** Compute $c^{(4),t} \leftarrow \mathrm{CRot}(\mathbf{C}_{\mathrm{rot},t}, c^{(3),t})$.
  7. **Homomorphic repacking:** Compute $c^{(5)} \leftarrow \mathrm{Repack}_{n_2}(\{c^{(4),t}\}_{t \in [T]}, \mathrm{pp}_{\mathrm{qck}})$.
  8. **Response compression:** Compute $\mathrm{ans} \leftarrow \mathrm{RespComp}(\mathrm{pp}_{\mathrm{rck}}, c^{(5)})$.
- Recover$(\mathrm{qk}, \mathrm{st}, \mathrm{ans}) \to \mathbf{d}$: Given the query key $\mathrm{qk} = (\mathbf{s}, S_1, S_2)$, the secret state $\mathrm{st} = \perp$ and the answer ans, output $\mathbf{d} \leftarrow \mathrm{RespCompRecover}(S_2, \mathrm{ans})$.

FIGURE 9. The VIA-B Protocol

MLWEs-to-RLWE conversion. For an integer d that is a power of 2 and d many (d,n)-MLWE ciphertexts $\mathbf{c}$, pairwise grouping followed by MLWEs-to-MLWE conversion yields $d/2$ many $(d/2, 2n)$-MLWE ciphertexts. Subsequent recursive application of this procedure over $\log d$ iterations ultimately produces a single $(1, nd)$-MLWE ciphertext, that is an RLWE ciphertext, denoted by $\mathrm{MLWEstoRLWE}(\mathbf{c}, \hat{\mathbf{c}}_{\mathrm{toRLWE}})$, where $\hat{\mathbf{c}}_{\mathrm{toRLWE}}$ is the key-switching keys used in the MLWEs-to-MLWE conversions.

Homomorphic repacking. Homomorphic repacking comprises two sequential operations: RLWE-to-MLWE extraction and MLWEs-to-RLWE conversion. For integers $d \le k \le n$ that are powers of 2. Given d many ciphertexts $\{c_i \leftarrow \mathrm{RLWE}_S(M_i)\}_{i \in [d]}$ with $M_i \in \mathcal{R}_{n,p}$, RLWE-to-MLWE extraction yields d many $(dn/k, k/d)$-MLWE ciphertexts, then MLWEs-to-RLWE conversion condenses the messages distributed across the d MLWE ciphertexts and $dn/k - d$ zero ciphertexts into a single RLWE ciphertext denoted by $\mathrm{Repack}_k(\{c_i\}_{i \in [d]}, \hat{\mathbf{c}}_{\mathrm{toRLWE}})$ of message $\iota_0^{k \to n} \iota^{k/d \to k}(\pi_0^{n \to k/d}(M_0), \ldots, \pi_0^{n \to k/d}(M_{d-1}))$, where $\hat{\mathbf{c}}_{\mathrm{toRLWE}}$ is the key-switching keys used in the MLWEs-to-MLWE conversions.

Remark. For all $k = d/2^m \le n$ with $m \ge 1$, if the secret key of the intermediate $(k, n/k)$-MLWE is selected as $\pi^{n \to n/k}(S_1)$, the public parameter $\hat{\mathbf{c}}_{\mathrm{toRLWE}}$ required for repacking becomes part of the LWE-to-RLWE conversion's public parameters. Consequently, no new public parameters need to be introduced in the repacking algorithm. We will adopt this configuration.

4.7. **VIA-B Protocol.** From an abstract perspective, VIA-B utilizes a query compression algorithm to compress multiple queries individually. Subsequently, the server executes the VIA-C response algorithm for each compressed query. The resulting answers are then packed into multiple RLWE

ciphertexts using the homomorphic repacking algorithm. If there are more than one packed ciphertext, the answers are vectorized employing the vectorization algorithms detailed in [14], thereby generating the final response. Comprehensive implementation details are provided in Figure 9.

5. Implementation and Evaluation

In this section, we describe our implementation and experimental evaluation of the proposed protocols.

5.1. Parameter Selection.

We choose the scheme parameters to tolerate a correctness error of at most 2^{-40}. Simultaneously, we choose the lattice parameters to ensure that each of the underlying RLWE/MLWE assumptions which we require for security provides 110 bits of classical security. We use the lattice estimator tool [46] for our security estimates. Here, we describe how we select the primary parameters of our scheme and list our parameter choices in Appendix B.

Lattice parameters. In VIA, we rely on two different RLWE assumptions:

- RLWE over the ring $\mathcal{R}_{n_1,q_1}$ with error distribution $\chi_{1,E}$ and secret key distribution $\chi_{1,S}$. The rotation ciphertext and the control bits are encrypted over $\mathcal{R}_{n_1,q_1}$. We set the ring dimension to be $n_1 = 2048$ and $q_1 = q_{1,1} * q_{1,2}$ to be a 57-bit modulus where $q_{1,1}, q_{1,2} = 1 \mod 2n_1$ are primes to support fast NTT evaluation. We take $\chi_{1,S}$ to be the uniform distribution on the interval $[-2, 2]$, and the error distribution $\chi_{1,E}$ is a discrete Gaussian distribution with standard deviation $\sigma_{1,E} = 1$.
- RLWE over the small ring $\mathcal{R}_{n_2,q_2}$ with error distribution $\chi_{2,E}$ and secret key distribution $\chi_{2,S}$. The ring-switching key and the select bits are encrypted over $\mathcal{R}_{n_2,q_2}$. We set the ring dimension to be $n_2 = 512$ and q_2 to be a 35-bit modulus where $q_2 = 1 \mod 2n_2$ is a prime to support fast NTT evaluation. We take $\chi_{2,S}$ and the error distribution $\chi_{1,E}$ to be discrete Gaussian distributions with standard deviation $\sigma_{1,E} = 4960$.

The VIA-C and VIA-B rely on MLWE assumptions. Since the (m, n)-MLWE problem is at least as hard as the RLWE problem of degree mn with the same error and secret key distributions, the LWE and MLWE parameters are selected to align with those defined for RLWE over ring $\mathcal{R}_{n_1,q_1}$. Consequently, we only give two RLWE assumptions:

- RLWE over the ring $\mathcal{R}_{n_1,q_1}$ with error distribution $\chi_{1,E}$ and secret key distribution $\chi_{1,S}$. We set the ring dimension to be $n_1 = 2048$ and $q_1 = q_{1,1} * q_{1,2}$ to be a 75-bit modulus where $q_{1,1}, q_{1,2} = 1 \mod 2n_1$ are primes to support fast NTT evaluation. We take $\chi_{1,S}$ and $\chi_{1,E}$ to be discrete Gaussian distributions with standard deviations $\sigma_{1,S} = 32$ and $\sigma_{1,E} = 1024$ respectively.
- RLWE over the small ring $\mathcal{R}_{n_2,q_3}$ with error distribution $\chi_{2,E}$ and secret key distribution $\chi_{2,S}$. We set the ring dimension to be $n_2 = 512$ and q_3 to be a 23-bit modulus where $q_3 = 1 \mod 2n_2$ is a prime to support fast NTT evaluation. We take $\chi_{2,S}$ and the error distribution $\chi_{2,E}$ to be discrete Gaussian distributions with standard deviation $\sigma_{1,E} = 26$.

Database configuration. The dimension of the reduced ring is set to $n_2 = 512$. As blinded extraction is adopted, the record size is $\log p$ bits, and database encoding in the Setup database phase is not required; only an NTT is performed per $n_1 = 2048$ elements. For VIA, the plaintext modulus is configured to $p = 256$, while for VIA-C and VIA-B, it is reconfigured to $p = 16$.

Gadget parameters. In VIA, different gadget parameters are selected for the two RLev ciphertexts within the RGSW ciphertexts of the DMux control bits and the CMux selection bits. The specific parameter sets are enumerated in Table 5 and Table 6 within Appendix B.

Modulus choice.

- In VIA, we choose modulus $q_1 = q_{1,1}q_{1,2}$ to be 57-bit modulus where $q_{1,1}, q_{1,2} = 1 \mod 2n_1$ are primes to support fast NTT evaluation. We implement arithmetic modulo q_1 using the Chinese remainder theorem. Similarly, we choose $q_2, q_3 = 1 \mod 2n_2$ to be a 35-bit prime and a 31-bit prime for fast NTT evaluation. Finally, we set $q_4 = 2^{15}$.
- In VIA-C and VIA-B, we choose modulus $q_1 = q_{1,1}q_{1,2}$ to be 75-bit modulus where $q_{1,1}, q_{1,2} = 1 \mod 2n_1$ are primes to support fast NTT evaluation. We implement arithmetic modulo q_1 using the Chinese remainder theorem. Similarly, we choose $q_2, q_3 = 1 \mod 2n_2$ to be a 34-bit prime and a 23-bit prime for fast NTT evaluation. Finally, we set $q_4 = 2^{12}$.

PRG compression. We use a standard optimization [24, 5, 22, 14, 20, 15] to reduce the query size, wherein the client sends a PRG seed in place of the random component of ciphertexts in the query.

5.2. **Experimental Evaluation of VIA and VIA-C.**

Experimental setup. Our implementation of VIA and VIA-C contains roughly 4,000 lines of C++. We use a workstation with AMD 9950X cpu, 128 GB of memory, and running Ubuntu 22.04.5 for our experiments. We use clang 19.1.7 as our C++ compiler. The processor supports the AVX-512DQ and AVX-512IFMA52 instruction sets and we enable SIMD instruction set support for all schemes. We use a single-threaded execution environment for all measurements. All measured running times were averaged over at least 5 trials and have a standard deviation of at most 5% of the average value.

Comparison schemes. We compare VIA against the state-of-the-art single-server PIR schemes without offline communication: HintlessPIR [22] and YPIR [20], and compare VIA-C against the state-of-the-art single-server PIR schemes with offline communication: SimplePIR [5] (high-throughput) and Respire [15] (communication-efficient).

Macrobenchmarks. Table 1 presents a comparative analysis of VIA and VIA-C protocol performance against other PIR protocols. Without offline communication, VIA achieves 690.25 KB online communication for a 32 GB database—3.7× reduction compared to YPIR and a 26.9× compared to HintlessPIR. This communication efficiency results from VIA's logarithmic communication complexity, whereas YPIR and HintlessPIR exhibit $O(\sqrt{N})$ complexity. Regarding the throughput, VIA exceeds HintlessPIR across 1 GB, 4 GB, and 32 GB databases, but marginally trails YPIR only in larger database sizes.

With offline communication, VIA-C achieves a 0.659 KB query for a 32 GB database—an 87.6× reduction relative to Respire. Simultaneously, it attains a 1.439 KB response, representing a 1.4× reduction. Total online communication is reduced 28.5× versus Respire. The query size reduction stems primarily from the proposed LWE-to-RLWE conversion algorithm, while the blinded extraction algorithm facilitates the response size reduction. Compared to SimplePIR, VIA-C optimizes online communication by 690× while maintaining throughput at the GB/s level.

Compared to lattice-based PIR protocols, VIA and VIA-C establish distinct throughput-communication trade-offs. For a 1 GB database, VIA reduces the total communication by a factor of 1.8 versus YPIR while sustaining marginally higher throughput. VIA-C exhibits a 28.5× reduction in communication compared to Respire for a 32 GB database, coupled with a 2.26× improvement in throughput. Although both VIA and VIA-C exhibit lower throughput than YPIR and SimplePIR at larger database, it is noteworthy that YPIR and SimplePIR necessitate significantly extended preprocessing time (YPIR requires 9.5× that of VIA; SimplePIR requires 50× that of VIA-C).

Preprocessing cost. SimplePIR, HintlessPIR, and YPIR necessitate substantial preprocessing time to attain efficient online computation performance. Conversely, Respire, VIA, and VIA-C each employ considerably more lightweight preprocessing protocols, consisting solely of database encoding and NTT operations. The accelerated preprocessing exhibited by both VIA and VIA-C, relative to Respire, stems directly from the integration of modulus switching prior to first-dimension processing. This optimization permits the execution of NTT operations under a small modulus. For a 32 GB database,

TABLE 1. Performance comparison of VIA and VIA-C with HintlessPIR [22], YPIR [20], SimplePIR [5], and Respire [15] under 1 GB, 4 GB, and 32 GB databases.

		Without Offline Communication			With Offline Communication		
Database	**Metric**	**HintlessPIR**	**YPIR**	**VIA**	**SimplePIR**	**Respire**	**VIA-C**
1 GB	**Offline Comm.**	–	–	–	128 MB	3.9 MB	14.8 MB
	Offline Comp.	119 s	12.18 s	1.06 s	94.1 s	33.75 s	2.09 s
	Query Size	453 KB	846 KB	473.1 KB	128 KB	7.66 KB	0.568 KB
	Response Size	3080 KB	12 KB	15.5 KB	128 KB	2 KB	1.439 KB
	Online Comp.	2.193 s	0.465 s	0.442 s	0.064 s	1.871 s	0.83 s
	Throughput	466.9 MB/s	2.15 GB/s	2.26 GB/s	15.63 GB/s	547.3 MB/s	1.2 GB/s
4 GB	**Offline Comm.**	–	–	–	256 MB	3.9 MB	14.8 MB
	Offline Comp.	426.9 s	41.937 s	4.195 s	374.3 s	137.49 s	7.786 s
	Query Size	584 KB	1230 KB	587.1 KB	256 KB	14.71 KB	0.604 KB
	Response Size	6160 KB	12 KB	15.5 KB	256 KB	2 KB	1.439 KB
	Online Comp.	4.196 s	0.986 s	1.361 s	0.222 s	6.273 s	2.777 s
	Throughput	976.2 MB/s	4.06 GB/s	2.94 GB/s	18.02 GB/s	652.96 MB/s	1.44 GB/s
32 GB	**Offline Comm.**	–	–	–	724 MB	3.9 MB	14.8 MB
	Offline Comp.	9252.3 s	315.231 s	33.34 s	3376.47 s	1101.33 s	67.539 s
	Query Size	1064 KB	2560 KB	674.75 KB	724 KB	57.77 KB	0.659 KB
	Response Size	17514 KB	12 KB	15.5 KB	724 KB	2 KB	1.439 KB
	Online Comp.	17.391 s	7.086 s	10.286 s	2.674 s	45.851 s	20.307 s
	Throughput	1.84 GB/s	4.52 GB/s	3.11 GB/s	11.97 GB/s	714.66 MB/s	1.58 GB/s

VIA necessitates merely 33.34 seconds of preprocessing time, while VIA-C requires only 67.539 seconds. These durations represent at least a 4.67× reduction of SimplePIR, HintlessPIR, and YPIR.

Throughput. Fig. 10 illustrates online computation time in VIA and VIA-C protocols as a function of database scale, benchmarked against YPIR and Respire—the current state-of-the-art schemes with and without offline communication, respectively. Without offline communication, VIA achieves higher throughput than YPIR for small databases but is slower than YPIR for large databases; nevertheless,

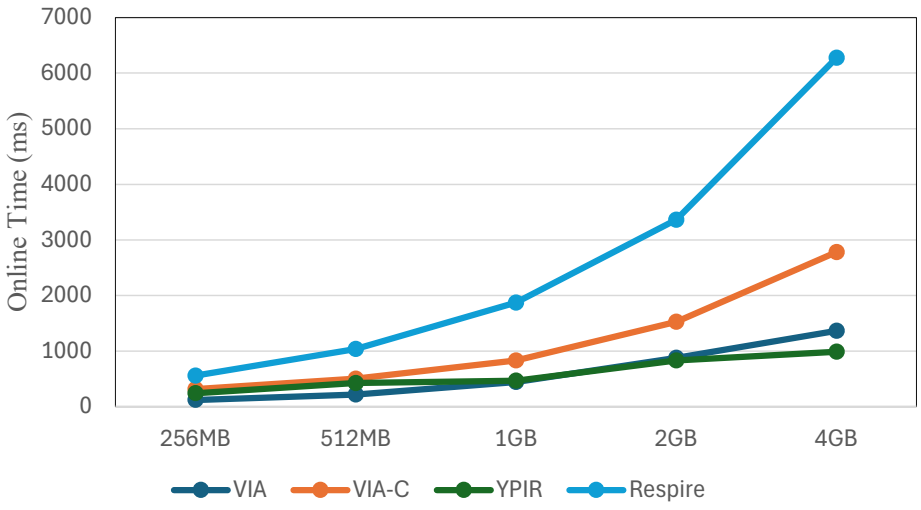

FIGURE 10. The Online Time of VIA, VIA-C, YPIR and Respire

it maintains GB/s throughput. With offline communication enabled, VIA-C achieves an approximate 2.6× improvement in throughput relative to Respire.

Microbenchmarks. Fig. 11 presents a fine-grained analysis of server computation time for the VIA, VIA-C, and Respire protocols. VIA-C's novel query compression algorithm, integrated with DMux operations, demonstrates a marginal performance improvement over Respire's corresponding compression technique during ciphertext generation. Crucially, First Dimension processing is the dominant computational overhead. VIA achieves speeds approximately 2× faster than VIA-C and 4× faster than Respire. The performance differential between VIA and VIA-C stems directly from their distinct plaintext modulus configurations: VIA employs a 256-bit modulus, whereas VIA-C utilizes a 16-bit modulus. This disparity necessitates twice the computational cost for VIA-C relative to VIA at equivalent database scales. Similarly, VIA-C's 2× reduction in First Dimension processing time compared to Respire is directly attributable to the aforementioned modulus switching pre-processing step.

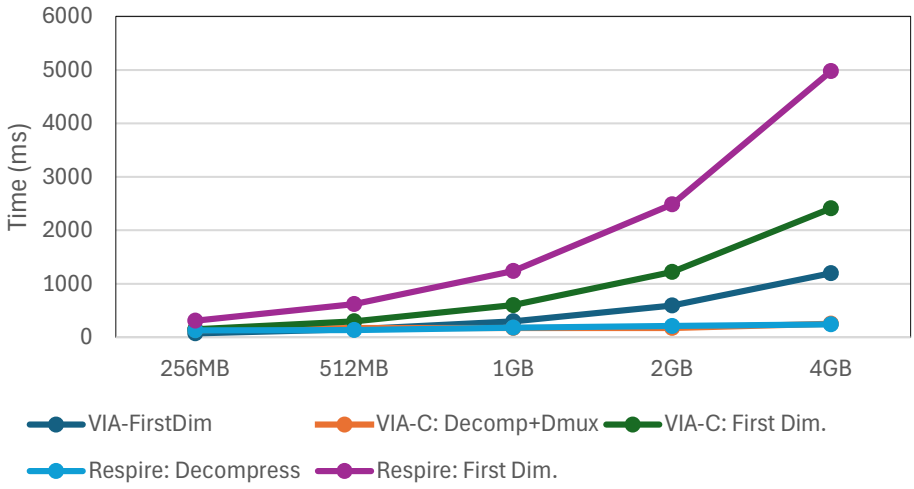

FIGURE 11. The Server Time of VIA, VIA-C and Respire

5.3. **Experimental Evaluation of VIA-B.** In this section, we compare VIA with Respire[15] in batch setting, called Respire-B.

Respire-B is implemented based on Respire, employing homomorphic repacking and vectorization algorithms. VIA-B utilizes our novel homomorphic repacking algorithm and Respire's vectorization technique built upon VIA-C. However, VIA-B eliminates vectorization entirely for batch sizes below 2048. As Section 5.2 previously detailed the online computation comparison between VIA-C and Respire, we focus exclusively here on quantifying the communication efficiency gains achieved by our novel query compression and homomorphic repacking algorithms within the batching framework.

Communications. Table 2 presents a comparative analysis of communication overhead for VIA-B against Respire-B. Due to the novel query compression algorithm, VIA-B consistently outperforms Respire-B in query size. For small record databases ($\leq$ 512-byte records), VIA-B also maintains superior response efficiency compared to Respire-B. Within tiny record databases (1-byte record), VIA-B achieves a 3.5× reduction in query size and a 127× reduction in response size relative to Respire-B. Furthermore, owing to substantial error accumulation inherent to Respire's homomorphic repacking algorithm, Respire-B exhibits limited support for tiny record databases.

TABLE 2. Comparison of VIA-B and batched Respire [15] for two different database configurations and batch sizes T. Each database record is 1 bytes.

		Batch Size T = 32		**Batch Size T = 256**	
Database	**Metric**	**Respire**	**VIA-B**	**Respire**	**VIA-B**
256 MB	**Offline Comm.**	4.6 MB	14.8 MB	4.6 MB	14.8 MB
	Query Size	67 KB	17 KB	326 KB	135.9 KB
	Response Size	31.8 KB	1.48 KB	234 KB	1.81 KB
1 GB	**Offline Comm.**	4.6 MB	14.8 MB	4.6 MB	14.8 MB
	Query Size	113 KB	18.16 KB	513 KB	145.31 KB
	Response Size	31.8 KB	1.48 KB	230 KB	1.81 KB

5.4. **Asymptotic Analysis.** Table 3 presents an asymptotic analysis of different PIR protocols. The asymptotic behavior of VIA-C is consistent with that of Respire; however, its constant factors are significantly smaller. As a result, VIA-C outperforms Respire in both communication and computational overhead. It is noteworthy that, among practical PIR protocols without offline communication, VIA is the first scheme to achieve $\widetilde{O}_\lambda(1)$ communication complexity.

TABLE 3. Asymptotic Analysis for PIR Protocols

Scheme	**Offline**		**Online**	
	Comp.	**Comm.**	**Comp.**	**Comm.**
Spiral [14]	$O_\lambda(N)$	$O_\lambda(1)$	$O_\lambda(N)$	$\widetilde{O}_\lambda(1)$
SimplePIR [5]	$O_\lambda(N)$	$O_\lambda(\sqrt{N})$	$O_\lambda(N)$	$O_\lambda(\sqrt{N})$
HintlessPIR [22]	$O_\lambda(N)$	–	$O_\lambda(N)$	$O_\lambda(\sqrt{N})$
YPIR [20]	$O_\lambda(N)$	–	$O_\lambda(N)$	$O_\lambda(\sqrt{N})$
Respire [15]	$O_\lambda(N)$	$O_\lambda(1)$	$O_\lambda(N)$	$\widetilde{O}_\lambda(1)$
VIA	$O_\lambda(N)$	–	$O_\lambda(N)$	$\widetilde{O}_\lambda(1)$
VIA-C	$O_\lambda(N)$	$O_\lambda(1)$	$O_\lambda(N)$	$\widetilde{O}_\lambda(1)$

6. Conclusion

We propose the VIA scheme, which maintains high throughput while requiring only $O(\log N)$ online communication overhead, completely eliminating the offline communication. We also design a novel LWE-to-RLWE conversion technique, significantly reducing noise growth. Leveraging this technique, we present the VIA-C scheme with offline communication, which further compresses the online query and download size. Both schemes have been implemented and tested, demonstrating high communication efficiency and outperforming state-of-the-art works.

References

[1] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 965–981, 1998.

[2] S. Angel and S. Setty, "Unobservable communication over fully untrusted infrastructure," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 551–569.

[3] P. Mittal, F. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg, "{PIR-Tor}: Scalable anonymous communication using private information retrieval," in *20th USENIX security symposium (USENIX security 11)*, 2011.

[4] N. Borisov, G. Danezis, and I. Goldberg, "Dp5: A private presence service," *Proceedings on Privacy Enhancing Technologies*, 2015.

[5] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, "One server for the price of two: Simple and fast {Single-Server} private information retrieval," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3889–3905.

[6] D. Kogan and H. Corrigan-Gibbs, "Private blocklist lookups with checklist," in *30th USENIX security symposium (USENIX Security 21)*, 2021, pp. 875–892.

[7] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish, "Scalable and private media consumption with popcorn," in *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, 2016, pp. 91–107.

[8] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: Single database, computationally-private information retrieval," in *Proceedings 38th annual symposium on foundations of computer science*. IEEE, 1997, pp. 364–373.

[9] C. Cachin, S. Micali, and M. Stadler, "Computationally private information retrieval with polylogarithmic communication," in *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer, 1999, pp. 402–414.

[10] A. Beimel, Y. Ishai, and T. Malkin, "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 Proceedings 20*. Springer, 2000, pp. 55–73.

[11] C. Gentry and Z. Ramzan, "Single-database private information retrieval with constant communication rate," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2005, pp. 803–815.

[12] S. Angel, H. Chen, K. Laine, and S. Setty, "Pir with compressed queries and amortized query processing," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 962–979.

[13] H. Chen, I. Chillotti, and L. Ren, "Onion ring oram: Efficient constant bandwidth oblivious ram from (leveled) tfhe," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 345–360.

[14] S. J. Menon and D. J. Wu, "Spiral: Fast, high-rate single-server pir via fhe composition," in *2022 IEEE symposium on security and privacy (SP)*. IEEE, 2022, pp. 930–947.

[15] A. Burton, S. J. Menon, and D. J. Wu, "Respire: High-rate pir for databases with small records," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 1463–1477.

[16] A. Davidson, G. Pestana, and S. Celi, "Frodopir: Simple, scalable, single-server private information retrieval," *Proceedings on Privacy Enhancing Technologies*, 2023.

[17] M. Zhou, A. Park, W. Zheng, and E. Shi, "Piano: extremely simple, single-server pir with sublinear server computation," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 4296–4314.

[18] A. Lazzaretti and C. Papamanthou, "Treepir: Sublinear-time and polylog-bandwidth private information retrieval from ddh," in *Annual International Cryptology Conference*. Springer, 2023, pp. 284–314.

[19] C. A. Melchor, J. Barrier, L. Fousse, and M.-O. Killijian, "Xpir: Private information retrieval for everyone," *Proceedings on Privacy Enhancing Technologies*, pp. 155–174, 2016.

[20] S. J. Menon and D. J. Wu, "{YPIR}:{High-Throughput}{Single-Server}{PIR} with silent preprocessing," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5985–6002.

[21] A. Henzinger, E. Dauterman, H. Corrigan-Gibbs, and N. Zeldovich, "Private web search with tiptoe," in *Proceedings of the 29th symposium on operating systems principles*, 2023, pp. 396–416.

[22] B. Li, D. Micciancio, M. Raykova, and M. Schultz-Wu, "Hintless single-server private information retrieval," in *Annual International Cryptology Conference*. Springer, 2024, pp. 183–217.

[23] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[24] M. H. Mughees, H. Chen, and L. Ren, "Onionpir: Response efficient single-server pir," in *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, 2021, pp. 2292–2306.

[25] D. Micciancio and J. Sorrell, "Ring packing and amortized fhew bootstrapping," *Cryptology ePrint Archive*, 2018.

[26] C. Boura, N. Gama, M. Georgieva, and D. Jetchev, "Chimera: Combining ring-lwe-based fully homomorphic encryption schemes," *Journal of Mathematical Cryptology*, vol. 14, no. 1, pp. 316–338, 2020.

[27] W.-j. Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu, "Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1057–1073.

[28] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (ring) lwe ciphertexts," in *International conference on applied cryptography and network security*. Springer, 2021, pp. 460–479.

[29] Y. Bae, J. H. Cheon, J. Kim, J. H. Park, and D. Stehlé, "Hermes: efficient ring packing using mlwe ciphertexts and application to transciphering," in *Annual International Cryptology Conference*. Springer, 2023, pp. 37–69.

[30] O. REGEV, "On lattices, learning with errors, random linear codes, and cryptography," *37th STOC, 2005*, pp. 84–93, 2005.

[31] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*. Springer, 2010, pp. 1–23.

[32] A. Ali, T. Lepoint, S. Patel, M. Raykova, P. Schoppmann, K. Seth, and K. Yeo, "{Communication–Computation} trade-offs in {PIR}," in *30th USENIX security symposium (USENIX Security 21)*, 2021, pp. 1811–1828.

[33] C. Gentry and S. Halevi, "Compressible fhe with applications to pir," in *Theory of cryptography conference*. Springer, 2019, pp. 438–464.
[34] J. Park and M. Tibouchi, "Shecs-pir: somewhat homomorphic encryption-based compact and scalable private information retrieval," in *European symposium on research in computer security*. Springer, 2020, pp. 86–106.
[35] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on computing*, vol. 43, no. 2, pp. 831–871, 2014.
[36] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Annual cryptology conference*. Springer, 2012, pp. 868–886.
[37] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.
[38] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*. Springer, 2013, pp. 75–92.
[39] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.
[40] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.
[41] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, "Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2021, pp. 670–699.
[42] C. Gentry, S. Halevi, C. Peikert, and N. P. Smart, "Ring switching in bgv-style homomorphic encryption," in *International Conference on Security and Cryptography for Networks*. Springer, 2012, pp. 19–37.
[43] K. Cong, D. Das, J. Park, and H. V. Pereira, "Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 563–577.
[44] K. Cong, D. Das, G. Nicolas, and J. Park, "Panacea: Non-interactive and stateless oblivious ram," in *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2024, pp. 790–809.
[45] R. Wang, Y. Wen, Z. Li, X. Lu, B. Wei, K. Liu, and K. Wang, "Circuit bootstrapping: faster and smaller," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2024, pp. 342–372.
[46] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.

A. Proof of Lemma 4.1 and Lemma 4.2

Lemma A.1. *Under the notations as Section 4.4, let θ_c be the variance of error in c, θ_{ks} be the variance of error introduced by key-switching and θ' be the variance of error in* $\mathrm{Conv}_k(c, \{\mathrm{ksk}^{(i)}\})$. *Then $\theta' \le \theta_c + d\theta_{\mathrm{ks}}$.*

Proof. View $c_{(i)}$ as an MLWE ciphertext of message $-A_{mi}S_{mi} - \cdots - A_{mi+m-1}S_{mi+m-1}$ under key $S_{(i)}$ without noise, and view $(\mathbf{0}, B)$ as a trivial MLWE ciphertext of $\langle \mathbf{A}, \mathbf{S}\rangle + M$ with the same noise e as c under any key. Then applying MLWE embedding and MLWE key switching, we have $\bar{c}_i = \mathrm{KeySwitch}(c^{(i)}, \mathrm{ksk}^{(i)})$ is an MLWE ciphertext of message $\iota_0(-A_{mi}S_{mi} - \cdots - A_{mi+m-1}S_{mi+m-1})$ under key $\mathbf{S}'$ with noise e_i, where e_i is the noise introduced by key switching with variance θ_{ks}. So the total noise of $\mathrm{Conv}_k(c, \{\mathrm{ksk}^{(i)}\})$ is $e + \sum_{i=0}^{d-1} e_i$, thus the final noise variance satisfies $\theta' \le \theta_c + d\theta_{\mathrm{ks}}$. □

Lemma A.2. *Let θ_c be the variance of error in c, θ_{ks} be the variance of error introduced by key-switching and θ' be the variance of error in* $\mathrm{LWEtoRLWE}(c)$. *Then $\theta' \le \theta_c + 2\theta_{\mathrm{ks}} \log n$.*

Proof. By Lemma 4.1, each MLWE-to-MLWE conversion introduces additional $2\theta_{\mathrm{ks}}$ noise variance. As LWE-to-RLWE conversion requires $\log n$ sequential MLWE-to-MLWE conversions, the cumulative noise variance adheres to $\theta' \le \theta_c + 2\theta_{\mathrm{ks}} \log n$. □

B. Parameters of VIA and VIA-C

In this section, we give the concrete database dimensions we use in VIA, VIA-C and VIA-B in Table 4, the database dimensions of VIA-B is the same as VIA-C. In all of our instantiations, we set $n_1 = 2048$, $n_2 = 512$. For VIA, $q_1 = 268369921 \cdot 268369921 \approx 2^{57}$, $q_2 = 34359214081 \approx 2^{35}$, $q_3 = 2147352577 \approx 2^{31}, q_4 = 2^{15}$ and $p = 256$. For VIA-C, $q_1 = 137438822401 \cdot 274810798081 \approx 2^{75}$, $q_2 = 17175674881 \approx 2^{34}$, $q_3 = 8380417 \approx 2^{23}, q_4 = 2^{12}$ and $p = 16$.

TABLE 4. Database dimensions I and J for VIA and VIA-C.

Database Size	VIA		VIA-C	
	I	J	I	J
1 GB	2^6	2^{13}	2^8	2^{12}
4 GB	2^8	2^{13}	2^9	2^{13}
32 GB	2^9	2^{15}	2^{11}	2^{14}

We also give the gadget parameters using in VIA, VIA-C and VIA-B in Table 5 and Table 6 respectively.

TABLE 5. The Gadget Parameters using in VIA.

Parameter	Gadget Length	Gadget Base
DMux	$(2, 2)$	$(370758, 370758)$
CMux	$(4, 3)$	$(24, 24)$
Ring-Switching Key	4	24

TABLE 6. The Gadget Parameters using in VIA-C and VIA-B.

Parameter	Gadget Length	Gadget Base
DMux	$(2, 2)$	$(55879, 55879)$
CMux	$(2, 2)$	$(81, 81)$
Conversion Key	18	18
Ring-Switching Key	8	8

C. Correctness and Security of VIA, VIA-C and VIA-B

In this section, we prove the correctness and security of the VIA, VIA-C and VIA-B protocol. Since VIA and VIA-C are sub-protocols of VIA-B, we focus exclusively on VIA-B in our correctness and security analysis.

C.1. **Security Analysis.** The security of VIA-B relies on a circular security or key-dependent message (KDM) security and MLWE assumption. Since VIA-B generates the batch queries using T independent queries of VIA-C, it suffices to show that VIA-C is secure.

Let $n_1, n_2, q_1, q_2, q_3, q_4, \chi_{1,E}, \chi_{1,S}, \chi_{2,E}, \chi_{2,S}$ be the lattice parameters used in VIA-B, and define the gadget parameters used in each of the underlying algorithms:

- Let $(\ell_{\mathrm{conv}}, B_{\mathrm{conv}})$ and $(\ell_{\mathrm{rs}}, B_{\mathrm{rs}})$ be the gadget parameters used in the conversion keys and ring-switching key, respectively.
- Let $(\ell_{\mathrm{ctrl},1}, \ell_{\mathrm{ctrl},2}, B_{\mathrm{ctrl},1}, B_{\mathrm{ctrl},2}), (\ell_{\mathrm{sel},1}, \ell_{\mathrm{sel},2}, B_{\mathrm{sel},1}, B_{\mathrm{sel},2})$ and $(\ell_{\mathrm{rot},1}, \ell_{\mathrm{rot},2}, B_{\mathrm{rot},1}, B_{\mathrm{rot},2})$ be the gadget length and gadget base parameters used in the DMux, CMux and CRot respectively.

We start by defining a sequence of hybrid experiments, each parameterized by a bit $b \in \{0, 1\}$.

1. $\mathrm{Hyb}_0^{(b)}$: The challenger samples $((\mathrm{pp}_{\mathrm{qck}}, \mathrm{pp}_{\mathrm{rck}}), \mathrm{qk}) \leftarrow \mathrm{Setup}(1^\lambda)$ and gives $\mathrm{pp} = (\mathrm{pp}_{\mathrm{qck}}, \mathrm{pp}_{\mathrm{rck}})$ to adversary $\mathcal{A}$. When adversary $\mathcal{A}$ makes a query on a pair of indices $(\mathrm{idx}_0, \mathrm{idx}_1)$, the challenger replies with $\mathrm{qu} \leftarrow \mathrm{Query}(\mathrm{qk}, \mathrm{idx}_b)$.
2. $\mathrm{Hyb}_1^{(b)}$: Same as $\mathrm{Hyb}_0^{(b)}$, except the challenger samples $\mathrm{pp}_{\mathrm{rck}} \leftarrow \mathcal{R}_{n_1,q_2}^{\ell_{\mathrm{rs}}}$.
3. $\mathrm{Hyb}_2^{(b)}$: Same as $\mathrm{Hyb}_1^{(b)}$, except the challenger samples $\mathrm{pp}_{\mathrm{qck}} = (\hat{c}_{\mathrm{toRLWE}}, \hat{c}_{\mathrm{toRGSW}}) \leftarrow \mathcal{R}_{n_1,q_1}^{\ell_{\mathrm{conv}}} \times \mathcal{R}_{n_1,q_1}^{\ell_{\mathrm{conv}}}$.

4. $\mathrm{Hyb}_3^{(b)}$: Same as $\mathrm{Hyb}_2^{(b)}$, except when response to each query, the challenger samples $\mathrm{qu} \leftarrow \mathbb{Z}_{q_1}^N$.

For an adversary $\mathcal{A}$, we write $\mathrm{Hyb}_i^{(b)}(\mathcal{A})$ to denote the output distribution of an execution of $\mathrm{Hyb}_i^{(b)}$ with adversary $\mathcal{A}$. Since the challenger's behavior in $\mathrm{Hyb}_3^{(b)}$ is independent of the bit b, we have that for all adversaries $\mathcal{A}$, $\mathrm{Hyb}_3^{(0)} \equiv \mathrm{Hyb}_3^{(1)}$. Thus, it suffices to show that each adjacent pair of distributions are computationally indistinguishable:

- $\mathrm{Hyb}_0^{(b)}(\mathcal{A})$ and $\mathrm{Hyb}_1^{(b)}(\mathcal{A})$ are computationally indistinguishable: The only difference between these experiments is the distribution of $\mathrm{pp}_{\mathrm{rck}}$. Under the MLWE assumption, we have that the distribution of $\mathcal{R}_{n_1,q_2}^{2\ell_{\mathrm{rs}}}$ and $\mathrm{RespCompSetup}(1^\lambda, S_1, S_2)$ are computationally indistinguishable.
- $\mathrm{Hyb}_1^{(b)}(\mathcal{A})$ and $\mathrm{Hyb}_2^{(b)}(\mathcal{A})$ are computationally indistinguishable: The only difference between these experiments is the distribution of $\mathrm{pp}_{\mathrm{qck}}$. Under the MLWE assumption, we have that the distribution of $\mathcal{R}_{n_1,q_1}^{\ell_{\mathrm{conv}}} \times \mathcal{R}_{n_1,q_1}^{\ell_{\mathrm{conv}}}$ and $\mathrm{QueryCompSetup}(1^\lambda, \mathbf{s}, S)$ are computationally indistinguishable.
- $\mathrm{Hyb}_2^{(b)}(\mathcal{A})$ and $\mathrm{Hyb}_3^{(b)}(\mathcal{A})$ are computationally indistinguishable: The only difference between these experiments is the distribution of qu. Under the MLWE assumption, we have that the distribution of $\mathbb{Z}_{q_1}$ and the message part of LWE ciphertext are computationally indistinguishable. So the distribution of $\mathbb{Z}_{q_1}$ and $\mathrm{Query}(\mathrm{qk}, \mathrm{idx}_b)$ are computationally indistinguishable.

Since each pair of adjacent distributions are computationally indistinguishable, query security holds.

C.2. **Correctness Analysis.** We begin by presenting the error estimates for CMux, DMux, query compression algorithms and response compression algorithms.

Lemma C.1 (CGGI [40])**.** *Let $c_i \leftarrow \mathrm{RLWE}_S(M_i)$ for $i = 0, 1$, encrypted select bit $C \leftarrow \mathrm{RGSW}_S(b)$ with $b \in \{0, 1\}$, and $c' \leftarrow \mathrm{CMux}(C, c_0, c_1)$. Assume that c_i has error variance θ_i, C has error variance θ_C, c' has error variance θ', and (ℓ_1, ℓ_2) and (B_1, B_2) are the gadget length and gadget base parameters of C. Then*

$$\theta' \leq \max(\theta_0, \theta_1) + n(\ell_1 B_1^2 + \ell_2 B_2^2)\theta_C/12 + nq^2/12B_1^{2\ell_1} + q^2/12B_2^{2\ell_2}.$$

Lemma C.2. *Let $c \leftarrow \mathrm{RLWE}_S(M)$, encrypted control bit $C \leftarrow \mathrm{RGSW}_S(b)$ with $b \in \{0, 1\}$, and $(c'_0, c'_1) \leftarrow \mathrm{DMux}(C, c)$. Assume that c has error variance θ_c, C has error variance θ_C, c'_i has error variance θ'_i, and (ℓ_1, ℓ_2) and (B_1, B_2) are the gadget length and gadget base parameters of C. Then*

$$\theta'_0 = \theta'_1 \leq \theta_c + n(\ell_1 B_1^2 + \ell_2 B_2^2)\theta_C/12 + nq^2/12B_1^{2\ell_1} + q^2/12B_2^{2\ell_2}.$$

Query Compression Algorithms. As illustrated in Figure 6 for the query compression algorithms, the following error estimates are provided. In the following, we will say that a ciphertext has error variance σ^2 if the error variance is at most σ^2.

Lemma C.3 (Wang et al. [45])**.** *Under the notations as Section 4.2, let θ_c be the the variance of error in $c = (A, B)$, θ_{toRGSW} be the variance of error in $\hat{c}_{\mathrm{toRGSW}}$, θ' be the variance of error in $\mathrm{RLWEtoRGSW}(c, \hat{c}_{\mathrm{toRGSW}})$ and θ_S be the variance of the secret key distribution. Let (ℓ, B) be the gadget parameter of $\hat{c}_{\mathrm{toRGSW}}$. Then*

$$\theta' \leq \frac{\ell B^2 \theta_{\mathrm{toRGSW}}}{12} + \frac{\|M\|_\infty^2 q^2}{12B^{2\ell}} + n\theta_S\theta_c,$$

where $\|M\|_\infty = \max_{i\in[n]} |M_i|$ is the infinity-norm of M.

Lemma C.4. *For all secret keys $\mathbf{s} \in \mathbb{Z}_{q_1}^n, S \in \mathcal{R}_{n,q_1}$ and for all $\mathbf{b}_{\mathrm{ctrl}} \in \{0,1\}^{\log I}$, $\mathbf{b}_{\mathrm{sel}} \in \{0,1\}^{\log J}$, $\mathbf{b}_{\mathrm{rot}} \in \{0,1\}^{\log[n_1/n_2]}$, assume that*

$$\begin{aligned} \mathrm{pp}_{\mathrm{qck}} &\leftarrow \mathrm{QueryCompSetup}(1^\lambda, \mathbf{s}, S) \\ \mathrm{qu} &\leftarrow \mathrm{QueryComp}(\mathbf{s}, \mathbf{b}_{\mathrm{ctrl}}, \mathbf{b}_{\mathrm{sel}}, \mathbf{b}_{\mathrm{rot}}) \\ \mathbf{C}_{\mathrm{ctrl}}, \mathbf{C}_{\mathrm{sel}}, \mathbf{C}_{\mathrm{rot}} &\leftarrow \mathrm{QueryDecomp}(\mathrm{pp}_{\mathrm{qck}}, \mathrm{qu}) \end{aligned}$$

and the LWE ciphertexts in qu *has error variance θ_c, the secret key S has variance θ_S, and $\hat{c}_{\mathrm{toRLWE}}$ and $\hat{c}_{\mathrm{toRGSW}}$ has error variances θ_{toRLWE} and θ_{toRGSW} respectively. Let (ℓ, B) be the gadget parameter used in* $\mathrm{pp}_{\mathrm{qck}}$*. Then*

- *Each RGSW ciphertext in* $\mathbf{C}_{\mathrm{ctrl},t}$ *have error variance*

$$\theta_{\mathrm{ctrl}} = \frac{\ell B^2 \theta_{\mathrm{toRGSW}}}{12} + \frac{q_1^2}{12B^{2\ell}} + n_1\theta_S(\theta_c + 2\theta_{\mathrm{ks}} \log n_1),$$

  where $\theta_{\mathrm{ks}} \leq n_1 \ell B^2 \theta_{\mathrm{toRLWE}}/12 + \theta_S q_1^2/12B^{2\ell}$.
- *Each RGSW ciphertext in* $\mathbf{C}_{\mathrm{sel}}$ *and* $\mathbf{C}_{\mathrm{rot}}$ *have error variance*

$$\theta_{\mathrm{sel}} = \theta_{\mathrm{rot}} = \theta_{\mathrm{ctrl}} q_2^2/q_1^2 + (1 + n_1\theta_S)/12.$$

Proof. By Lemma 4.2, the output ciphertext of LWEtoRLWE algorithm has error variance

$$\theta' = \theta_c + 2\theta_{\mathrm{ks}} \log n_1,$$

and by [45] Lemma 1, we have $\theta_{\mathrm{ks}} \leq \ell^2 B^2 \theta_{\mathrm{toRLWE}}/12 + \theta_S q_1^2/12B^{2\ell}$.

Then by Lemma C.3 and the message is 0 or 1, the output ciphertext of RLWEtoRGSW has error variance

$$\theta'' = \frac{\ell B^2 \theta_{\mathrm{toRGSW}}}{12} + \frac{q_1^2}{12B^{2\ell}} + n_1\theta_S\theta'.$$

Apply modulus-switching to those ciphertexts, the output ciphertext has error variance

$$\theta''' = \theta'' q_2^2/q_1^2 + (1 + n_1\theta_S)/12.$$

Since $\mathbf{C}_{\mathrm{ctrl}}$ is a combination of RLWE ciphertexts with error variance θ'', $\mathbf{C}_{\mathrm{ctrl}}$ has error variance θ''. Since $\mathbf{C}_{\mathrm{sel}}, \mathbf{C}_{\mathrm{rot}}$ are combinations of RLWE ciphertexts with error variance θ''', $\mathbf{C}_{\mathrm{sel}}, \mathbf{C}_{\mathrm{rot}}$ have error variance θ'''. □

Response Compression Algorithms. As illustrated in Figure 7 for the response compression algorithms, the following error estimates are provided.

Lemma C.5. *For all secret keys* $S_1 \in \mathcal{R}_{n_1,q_3}, S_2 \in \mathcal{R}_{n_2,q_3}$ *and for all ciphertext* $c \leftarrow \mathrm{RLWE}_{S_1}(M) \in \mathcal{R}^2_{n_1,q_2}$*, assume that*

$$\mathrm{pp}_{\mathrm{rck}} \leftarrow \mathrm{RespCompSetup}(1^\lambda, S_1, S_2)$$
$$\mathrm{ans} \leftarrow \mathrm{RespComp}(\mathrm{pp}_{\mathrm{rck}}, c)$$

and the ciphertexts c, ans *has error variance* θ_c *and* θ_{ans} *respectively, the secret key* S_1 *has variance* θ_{S_1}*, and* (ℓ, B) *is the gadget parameter used in* $\mathrm{pp}_{\mathrm{rck}}$*. Then*

$$\theta_{\mathrm{ans}} \leq \frac{\theta_c q_3^2}{q_2^2} + \frac{(1 + n_1\theta_{S_1})}{12} + \frac{\ell B^2 \theta_{\mathrm{rck}}}{12} + \frac{\theta_{S_1} q_3^2}{12B^{2\ell}}.$$

Proof. Apply modulus-switching to c, the output ciphertext has error variance

$$\theta' \leq \frac{\theta_c q_3^2}{q_2^2} + \frac{(1 + n_1\theta_{S_1})}{12}.$$

By [45] Lemma 1, the output ciphertext of ring-switching algorithm has error variance

$$\theta'' \leq \theta' + \frac{\ell B^2 \theta_{\mathrm{rck}}}{12} + \frac{\theta_{S_1} q_3^2}{12B^{2\ell}}.$$ □

Proof of Correctness. Assume the database setting as in Subsection 4.5. Let N be the number of database records and $T \leq n_1/n_3$ be the batch size (For cases where the $T > n_1/n_3$, VIA-B employs the same vectorization technique as Respire [15], which we do not discuss). Suppose that $\chi_{1,E}, \chi_{2,E}, \chi_{1,S}, \chi_{2,S}$ has variances $\theta_{1,E}, \theta_{2,E}, \theta_{1,S}, \theta_{2,S}$ respectively. Define the gadget parameters as Section C.1.

We now show that with high probability for a given $t \in [T]$, the decoded response d_t satisfies $d_t = X_t$. Specifically, we analyze the variance of the error in the ciphertext after each step of the answer algorithm.

1. **Query decompression:** Let

$$\mathbf{C}_{\mathrm{ctrl},t}, \mathbf{C}_{\mathrm{sel},t}, \mathbf{C}_{\mathrm{rot},t} \leftarrow \mathrm{QueryDecomp}(\mathrm{pp}_{\mathrm{qck}}, \mathrm{qu}_t)$$

   be the output of the query decompression algorithm. By Theorem C.4, the following holds:

- Each RGSW ciphertext in $\mathbf{C}_{\mathrm{ctrl},t}$ have error variance

$$\theta_{\mathrm{ctrl}} = \frac{\ell_{\mathrm{conv}} B_{\mathrm{conv}}^2 \theta_{1,E}}{12} + \frac{q_1^2}{12 B_{\mathrm{conv}}^{2\ell_{\mathrm{conv}}}} + n_1 \theta_{1,S}(\theta_{1,E} + 2\theta_{\mathrm{ks}} \log n_1),$$

  where $\theta_{\mathrm{ks}} \leq n_1 \ell_{\mathrm{conv}} B_{\mathrm{conv}}^2 \theta_{1,E}/12 + \theta_{1,S} q_1^2 / 12 B_{\mathrm{conv}}^{2\ell_{\mathrm{conv}}}$.
- Each RGSW ciphertext in $\mathbf{C}_{\mathrm{sel}}$ and $\mathbf{C}_{\mathrm{rot}}$ have error variance

$$\theta_{\mathrm{sel}} = \theta_{\mathrm{rot}} = \theta_{\mathrm{ctrl}} q_2^2 / q_1^2 + (1 + n_1 \theta_{1,S})/12.$$

2. **Homomorphic demultiplexer:** Let

$$(c_0^{(0),t}, c_1^{(0),t}, \ldots, c_{I-1}^{(0),t}) \leftarrow \mathrm{DMux}(\mathbf{C}_{\mathrm{ctrl},t}, \lfloor q/p \rfloor)$$

   be the output of the DMux algorithm. By Theorem C.1, every above RLWE ciphertext has error variance

$$\theta_{\mathrm{dmux}} = \theta_{\mathrm{ctrl}} + \theta_{\mathrm{DMux}} \log I,$$

   where

$$\theta_{\mathrm{DMux}} = \frac{n_1(\ell_{\mathrm{ctrl},1} B_{\mathrm{ctrl},1}^2 + \ell_{\mathrm{ctrl},2} B_{\mathrm{ctrl},2}^2)\theta_{ctrl}}{12} + \frac{n_1 q_1^2}{12 B_{\mathrm{ctrl},1}^{2\ell_{\mathrm{ctrl},1}}} + \frac{q_1^2}{12 B_{\mathrm{ctrl},2}^{2\ell_{\mathrm{ctrl},2}}}.$$

3. **Modulus switching** Let

$$c_i^{(1),t} \leftarrow \mathrm{ModSwitch}_{q_2,q_2}(c_i^{(0),t})$$

   be the output of the modulus-switching algorithm. Every above RLWE ciphertext has error variance

$$\theta_{\mathrm{ms}} = \theta_{\mathrm{dmux}} q_2^2 / q_1^2 + (1 + n_1 \theta_{1,S})/12.$$

4. **First dimension:** Let

$$c_j^{(2),t} = \sum_{i \in [I]} c_i^{(1),t} \, \mathrm{db}[i,j]$$

   be the output of the first-dimension algorithm. Every above RLWE ciphertext has error variance

$$\theta_{\mathrm{first}} = I n_1 \theta_{\mathrm{ms}} p^2 / 4.$$

5. **Homomorphic multiplexer:** Let

$$c^{(3),t} \leftarrow \mathrm{CMux}(\mathbf{C}_{\mathrm{sel},t}, (c_j^{(2),t})_{j \in J})$$

   be the output of the CMux algorithm. Every above RLWE ciphertext has error variance

$$\theta_{\mathrm{cmux}} = \theta_{\mathrm{first}} + \theta_{\mathrm{CMux}} \log J,$$

   where

$$\theta_{\mathrm{CMux}} = \frac{n_1(\ell_{\mathrm{sel},1} B_{\mathrm{sel},1}^2 + \ell_{\mathrm{sel},2} B_{\mathrm{sel},2}^2)\theta_{\mathrm{sel}}}{12} + \frac{n_1 q_2^2}{12 B_{\mathrm{sel},1}^{2\ell_{\mathrm{sel},1}}} + \frac{q_2^2}{12 B_{\mathrm{sel},2}^{2\ell_{\mathrm{sel},2}}}.$$

6. **Homomorphic rotation:** Let

$$c^{(4),t} \leftarrow \mathrm{CRot}(\mathbf{C}_{\mathrm{rot},t}, c^{(3),t})$$

   be the output of the CRot algorithm. Every above RLWE ciphertext has error variance

$$\theta_{crot}^2 = \theta_{\mathrm{cmux}} + \log(n_1/n_3)\theta_{\mathrm{CRot}},$$

   where

$$\theta_{\mathrm{CRot}} = \frac{n_1(\ell_{\mathrm{rot},1} B_{\mathrm{rot},1}^2 + \ell_{\mathrm{rot},2} B_{\mathrm{rot},2}^2)\theta_{\mathrm{rot}}}{12} + \frac{n_1 q_2^2}{12 B_{\mathrm{rot},1}^{2\ell_{\mathrm{rot},1}}} + \frac{q_2^2}{12 B_{\mathrm{rot},2}^{2\ell_{\mathrm{rot},2}}}.$$

7. **Homomorphic repacking:** Let
$$c^{(5)} \leftarrow \mathrm{Repack}(\{c^{(4),t}\}_{t\in[T]}, \mathrm{pp}_{\mathrm{qck}})$$
be the output of the repacking algorithm. By the similar proof of Lemma 4.2, the RLWE ciphertext has error variance
$$\theta_{\mathrm{rep}} = \theta_{\mathrm{crot}} + 2\theta_{\mathrm{ks}} \log n_2.$$
8. **Response Compress:** Let
$$\mathrm{ans} \leftarrow \mathrm{RespComp}(\mathrm{pp}_{\mathrm{rck}}, c^{(5)})$$
be the final answer. By Lemma C.5, the RLWE ciphertext ans has error variance
$$\theta_{\mathrm{ans}} = \frac{\theta_{\mathrm{rep}} q_3^2}{q_2^2} + \frac{(1 + n_1\theta_{1,S})}{12} + \frac{\ell_{\mathrm{rs}} B^2 \theta_{1,E}}{12} + \frac{\theta_{1,S} q_3^2}{12 B_{\mathrm{rs}}^{2\ell_{\mathrm{rs}}}}.$$

To ensure the correctness of the results, it is sufficient to guarantee that the final error
$$\|E_{\mathrm{final}}\|_\infty \le \lfloor (q_3 - q_4)/p \rceil/2 - 1.$$

Thus, we conclude that
$$\begin{aligned}\Pr(d_t = X_t : t \in [T]) &\ge \Pr(\|E_{\mathrm{final}}\|_\infty \ge \lfloor (q_3 - q_4)/p \rceil/2 - 1) \\ &\ge \mathrm{erfc}\left(\frac{\lfloor (q_3 - q_4)/p \rceil/2 - 1)}{\sqrt{2\theta_{\mathrm{ans}}}}\right).\end{aligned}$$

In our evaluation of VIA-B, we select the parameters such that the error rate is less than 2^{-40}.