RotPIR: Sublinear Single-Server PIR with Efficient Pre-Processing via Oblivious Homomorphic Rotation

Zhou Zhang* Song Bian† Zhenyu Guan‡ Giuseppe Persiano§

June 29, 2026

Abstract

Single-server PIR is a powerful cryptographic primitive that allows a client to retrieve public data from a remote server without revealing which item it accessed. Its privacy guarantee, however, inherently forces the server to do work linear in the database size, which is impractical at large scale. Client-Preprocessing PIR (CP-PIR) [Patel et al., CCS 2018] suggests to push linear computation to an offline preprocessing phase, enabling online queries with sublinear server computation and communication. While CP-PIR has since attracted considerable attention, designing an efficient preprocessing phase remains challenging. The original construction, along with much subsequent work (e.g., [Zhou et al., S&P 2024]), streams the entire database, which incurs linear communication that scales only to moderately sized data. A different approach consists of implementing the preprocessing using Fully Homomorphic Encryption (FHE) (e.g., [Corrigan-Gibbs et al., Eurocrypt 2022]), but the current best proposal (ThorPIR [Fisch et al., CCS 2024]) still imposes prohibitively expensive homomorphic computation on the server.

In this paper, we tackle the problem of designing a practical preprocessing phase for CP-PIR. We introduce RotPIR, a novel CP-PIR with a preprocessing that is simultaneously sublinear-communication (i.e., no database streaming) and computation-efficient, while keeping online communication and computation sublinear. RotPIR is obtained by co-designing our PIR protocol with the underlying FHE operations. Specifically, we propose an innovative preprocessing framework with an advanced homomorphic operator, *oblivious homomorphic rotation*, which enables the database to be rotated according to an encrypted offset, and significantly accelerate the offline server-side computation. Our experimental results show that RotPIR can achieve up to 1000× speedup in offline server-side computation when compared to the state-of-the-art CP-PIR protocol that do not stream the entire database during the preprocessing.

*Beihang University, China, zhouzhang@buaa.edu.cn
†Beihang University, China, sbian@buaa.edu.cn
‡Beihang University, China, guanzhenyu@buaa.edu.cn
§Università di Salerno, Italy, giuper@gmail.com

Contents

1 Introduction **3**
- 1.1 Our Contribution . . . 3
- 1.2 Technical Overview . . . 4
- 1.3 Related Works . . . 5

2 Preliminaries **7**
- 2.1 Private Information Retrieval . . . 7
- 2.2 Homomorphic Encryption . . . 8
  - 2.2.1 Ciphertext Types . . . 9
  - 2.2.2 Homomorphic Operators . . . 9

3 The RotPIR Protocol **10**
- 3.1 Protocol Overview . . . 10
- 3.2 Correctness and Security . . . 12

4 Handling Multiple Queries **13**
- 4.1 Protocol Overview . . . 14
- 4.2 Correctness and Security . . . 14

5 Oblivious Homomorphic Rotation **16**
- 5.1 Encrypted Rotation . . . 16
- 5.2 Multi-block and Packed Oblivious Rotations . . . 17
- 5.3 Query Compression . . . 20

6 Evaluation **20**
- 6.1 Microbenchmarks . . . 20
- 6.2 PIR Benchmarks . . . 21

7 Conclusion **23**

A Details and Proofs for RotPIR **29**
- A.1 Correctness . . . 29
- A.2 Security . . . 29

B Details and Proofs for mRotPIR **30**
- B.1 Correctness . . . 31
- B.2 Security . . . 33

C Full Notations, Ciphertext Type, and Operators **34**
- C.1 Noise Analysis of the Operator . . . 34

D Details of Oblivious Homomorphic Rotation **35**
- D.1 Additional Lemma and Proofs . . . 35
- D.2 Query Compression . . . 37

1 Introduction

Single-Server Private Information Retrieval (PIR) [38] allows a client to fetch a record from a public database stored on a single server without revealing which specific record was retrieved. Over the past quarter century, PIR has been applied to various applications, including private database search [63, 72, 34, 44], metadata-hiding messaging [57, 40, 8, 7], certificate transparency auditing [35, 56], password breach checking [46, 69, 5], caller ID lookup [9], among others. Despite its broad applicability, an inherent limitation of the single-server PIR is that the server must touch the entire database for each retrieval to preserve privacy [11]. As a result, classical single-server PIR protocols [54, 7, 58, 55, 35] still suffer from linear computational costs in the database size, which limits the practicality of such protocols in a large-scale database.

Patel et al. [62] suggested to allow the client to preprocess the database in order to reduce the computational burden on the server and Corrigan-Gibbs and Kogan [21] were the first to achieve sublinear server computation within this framework. Following these seminal works, other researchers [37, 20, 75, 76, 31, 42, 26, 36, 73, 10, 60] have proposed various PIR protocols leveraging offline client-preprocessing to break the online linear computation barrier on the server side. Specifically, in a client-preprocessing PIR (CP-PIR) protocol, the client first derives a set of private hints in the offline phase. From a high-level perspective, each hint can be viewed as an aggregated value of a subset of the database. Next in the online phase, the client leverages the hints to retrieve records privately with sublinear server computation and communication, making each online query efficient after preprocessing. However, the construction of the hints in the single-server setting requires the client to download the entire database in the offline phase to compute the hints locally [76, 42, 36, 73], which incurs a linear offline communication cost and becomes inefficient for massive databases.

To address the limitations of linear offline communication, a line of recent works [20, 75, 41, 26] explores how fully homomorphic encryption (FHE) can be used to construct single-server CP-PIR. The core idea is to let the server evaluate the hints homomorphically without learning which data entries the hints correspond to. ThorPIR [26] is the state-of-the-art (SOTA) protocol along this line of work. The result achieved in [26] is exciting as it shows that, at least in principle, this approach could lead to practical single-server CP-PIR with sublinear offline communication. However, ThorPIR [26] achieved this at the expense of intensive server computation. For instance, on a 360 GB database, ThorPIR [26] requires more than 10000 hours of server-side computation on a single GPU to generate the hint in the offline phase. Therefore, the following research question for single-server CP-PIR is still unanswered

> *can we design a* concretely efficient *single-server PIR protocol that achieves* sublinear *online server computation* without *streaming the entire database in the offline phase?*

1.1 Our Contribution

In this work, we continue along the line of research that uses FHE to reduce the communication requirement of the offline phase in CP-PIR. Our goal is to improve the practicality of the offline phase by avoiding linear database downloads without introducing prohibitive server-side FHE computation. We first present *RotPIR*, a novel single-server CP-PIR protocol for the single-query setting that achieves both sublinear offline communication and highly efficient offline server computation. The key idea is to co-design the PIR preprocessing with the underlying FHE operations. Instead of evaluating a general hint construction circuit under FHE, RotPIR introduces an innovative offline preprocessing framework with homomorphic operators tailored to hint construction, which significantly reduces the server-side offline computational overhead.

We further extend RotPIR to *mRotPIR*, a CP-PIR protocol that supports multiple queries after one preprocessing phase. mRotPIR amortizes the preprocessing cost across queries and reduces the per-query offline overhead compared with RotPIR. Asymptotically, mRotPIR achieves sublinear offline communication while keeping sublinear online communication and computation (see Table 1). In terms of concrete performance, mRotPIR achieves a 1000× speedup in offline server computation compared to ThorPIR [26], the SOTA CP-PIR with sublinear offline communication. Moreover, when considering the monetary cost in the

offline phase, mRotPIR achieves a 2.9×–3.8× reduction compared with CP-PIR protocols [76, 42, 36, 73] that require downloading the entire database during preprocessing.

In summary, the main contributions of this paper are:

- **Protocol Design:**
  - We introduce *RotPIR*, a single-server CP-PIR protocol for the single-query setting with sublinear offline communication and sublinear online server computation and communication (Section 3).
  - We extend RotPIR to *mRotPIR*, a multi-query CP-PIR protocol based on a chunk-based preprocessing stragegy. Thanks to a tighter analysis of the failure probability, mRotPIR requires fewer precomputed hints per chunk than prior work [20] for the same correctness guarantee, thereby reducing the concrete offline preprocessing overhead (Section 4).
- **Advanced Homomorphic Operator:**
  - We introduce a novel operator, *oblivious homomorphic rotation*, which serves as a key primitive for offline preprocessing and could be of independent interest to the FHE community (Section 5).
- **Implementation and Evaluation:**
  - We provide an open-source proof-of-concept implementation of RotPIR, including both CPU and GPU backends, and use it to conduct a thorough experimental evaluation of its cryptographic components and PIR performance (Section 6).

1.2 Technical Overview

We now provide a high-level overview of the core technical ideas behind our protocol.

Challenge 1: Generating random sets with FHE. Our starting point is the client-preprocessing PIR blueprint of Corrigan-Gibbs and Kogan [21]. In the protocol, the client privately retrieves multiple hints in the offline phase, each consisting of a random subset S of entries of the database $\mathbf{D}$ along with the corresponding parity $h_S = \oplus_{i \in S} \mathbf{D}[i]$. In the online phase, suppose the client wishes to retrieve the entry at index $\mathsf{idx} \in [N]$. The client identifies a hint where $\mathsf{idx} \in S$ and requests the parity of the records $r_{S \setminus \{\mathsf{idx}\}} = \oplus_{i \in S \setminus \{\mathsf{idx}\}} \mathbf{D}[i]$. The target record is then recovered as $\mathbf{D}[\mathsf{idx}] = h_S \oplus r_{S \setminus \{\mathsf{idx}\}}$. To ensure the privacy, the hints obtained in the offline phase must remain hidden from the online server. Existing single-server protocols [20, 76, 31, 42, 26, 36, 73, 10] achieve this either by streaming the entire database to the client for local parity computation, or leveraging FHE to homomorphically generate random subsets and compute parities entirely on the server. On the one hand, streaming the database incurs $\Omega(N)$ communication, which results in prohibitive communication overhead. On the other hand, current FHE-based techniques to generate random subsets, whether achieved via shuffling [26] or permutation network [58], involve large circuit depths and are highly inefficient in practice.

Insight 1: Replacing random sets with rotation. Our key insight is that we do not try to generate random subsets of the database completely, which requires shuffling or permuting. Instead, we demonstrate that a carefully designed partial shuffling suffices for secure hint construction. The conceptual design is presented in Figure 1. Our construction builds on the sublinear PIR protocols proposed in [42, 26]. However, we depart from these prior works by introducing a distinct and more computationally efficient strategy for offline hint evaluation. We take a 16-entry database $\mathbf{D}$ organized in a 4×4 matrix as an example. In the offline phase, the client sends four encrypted rotation offsets $\delta_0, \ldots, \delta_3$, together with an encrypted random offset r to the server. Instead of fully randomizing each row of the database, the server applies oblivious homomorphic rotations to each database row. It then homomorphically computes the column-wise parities of the rotated database, producing a ciphertext that encrypts the hint vector (h_0, h_1, h_2, h_3). In addition, using the encrypted offset r, the server computes an encrypted auxiliary hint $\mathbf{z} = (\mathbf{D}[0, r], \mathbf{D}[1, r], \mathbf{D}[2, r], \mathbf{D}[3, r])$. In the online phase, suppose the client wants to retrieve the item at index $\mathsf{idx} = (1, 3)$. Here, the stored hint h_2 corresponds to the target entry $\mathbf{D}[\mathsf{idx}]$, namely $h_2 = \mathbf{D}[0, 0] \oplus \mathbf{D}[1, 3] \oplus \mathbf{D}[2, 2] \oplus \mathbf{D}[3, 1]$. To preserve privacy, the client substitutes the index of the target row (row 1 in this case) with the value r and forms a

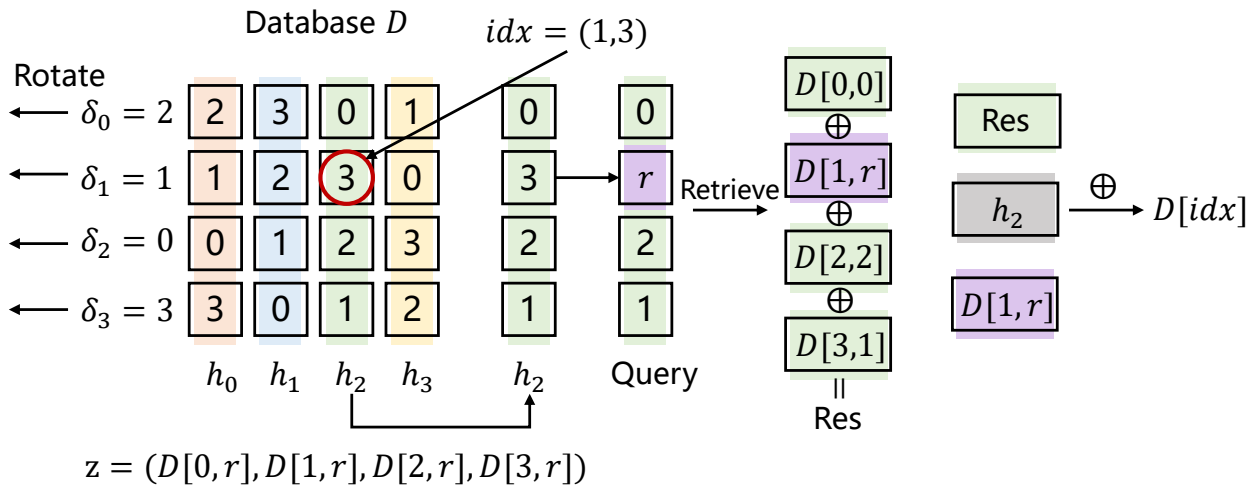


Figure 1: The conceptual illustrations of RotPIR.

query $(0, r, 2, 1)$ that is sent to the server. The server returns a single aggregated response $\mathsf{res} = \mathbf{D}[0,0] \oplus \mathbf{D}[1,r] \oplus \mathbf{D}[2,2] \oplus \mathbf{D}[3,1]$. The client then computes $h_2 \oplus z_1 \oplus \mathsf{res}$, and reveals the desired item $\mathbf{D}[1,3]$.

We provide a detailed description of the protocol, along with formal proofs showing that the rotation mechanism suffices to guarantee query privacy in Section 3.

Challenge 2: Supporting multiple queries. The construction above supports only single online query after each offline preprocessing phase. To address this limitation, Corrigan-Gibbs et al. [20] propose a general framework for extending single-query CP-PIR to the multi-query setting by partitioning the database into chunks and precomputing several independent hints for each chunk. Each online query then consumes one unused hint from the chunk containing the requested item. However, the concrete efficiency of this approach depends critically on the number of hints provisioned per chunk. To guarantee correctness, the framework of [20] generates a large number of hints per chunk to ensure that no chunk exhausts during the online queries, which substantially increases the offline preprocessing overhead.

Insight 2: Tighter bounds for multi-query preprocessing. To address this challenge, we follow the chunk-based framework of [20], but improve the concrete efficiency through a tighter analysis of the failure probability. As it turn out, we can use fewer precomputed hints per chunk under the same correctness guarantee, thereby reducing the concrete offline preprocessing overhead.

Challenge 3: Oblivious Rotation. Our protocol requires the server to homomorphically rotate the database without learning the rotation offset. However, achieving oblivious homomorphic rotation within FHE is non-trivial. Existing homomorphic rotation [29] typically assume the rotation offset to be known in plaintext, which directly conflicts with the requirement in RotPIR. Although blind rotations [24, 19] have been proposed to address this limitation, these approaches inherently involve negacyclic rotation and therefore cannot be directly applied in our construction.

Insight 3: Encrypted-index Homomorphic Rotation. To achieve oblivious homomorphic rotation, our key idea is to encrypt the rotation offset as a set of encrypted bits and iteratively use each bit to control whether a power-of-two plaintext homomorphic rotation is applied. Once all encrypted bits have been processed, the ciphertext is rotated to the target position in an entirely oblivious manner. Further cryptographic details and optimizations of oblivious homomorphic rotation are provided in Section 5.

1.3 Related Works

Early single-server PIR protocols. Following the blueprint of [39], early single-server PIR constructions [15, 48] were built upon the number-theoretic assumptions such as Paillier [61] or Damgård-Jurik [22]

Table 1: Asymptotic Comparisons Between Existing Client-Preprocessing Single-Server PIR Protocols.

Protocol		CHK [20]	ZLTS [75]	Piano [76]	RMS [65]	Plinko [36]	ThorPIR [26]	RotPIR	mRotPIR
Offline	Server Time	$\tilde{O}_\lambda(N)$	$\tilde{O}_\lambda(N)$	N/A	N/A	N/A	$\tilde{O}_\lambda(N)$	$\tilde{O}_\lambda(N)$	$\tilde{O}_\lambda(N)$
	Comm.	$\tilde{O}_\lambda(\sqrt{N})$	$\tilde{O}_\lambda(\sqrt{N})$	$O(N)$	$O(N)$	$O(N)$	$O_\lambda(N^{2/3})$	$O_\lambda(\sqrt{N})$	$O_\lambda(N^{2/3})$
Online	Server Time	$O(\sqrt{N})$	$\tilde{O}_\lambda(\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$	$O(N^{1/3})$	$O(\sqrt{N})$	$O(N^{2/3})$
	Comm.	$O(\sqrt{N})$	$\tilde{O}_\lambda(1)$	$O(\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$	$O(N^{1/3})$	$O(\sqrt{N})$	$O(N^{1/3})$
	# Queries	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$N^{1/3}$	1	$N^{1/3}$
Free-streaming		✓	✓	✗	✗	✗	✓	✓	✓
Concretely Efficiency		Low	Low	High	High	High	Low	High	High

encryption schemes. However, as observed in [67], the significant computational and communication overhead introduced by these protocols limit their practicality in large-scale datasets.

Practical linear single-server PIR protocols. Recent practical single-server PIR protocols [54, 7, 16, 27, 5, 55, 59, 23, 35, 34, 45, 56] are primarily built upon lattice-based cryptography, in particular the learning-with-errors (LWE) [64] and ring-learning-with-errors (RLWE) [52] assumptions. The main benefit of these protocols lies in their use of high-rate homomorphic ciphertexts [13, 25, 55], which enable more compact query and response encodings and thus reduce communication overhead. Nevertheless, a fundamental drawback is that the server-side computation scales linearly with the database size for each query, which results in significant computational overhead in large-scale datasets.

Sublinear single-server PIR protocols. Another line of work [11, 12, 62, 20, 76, 31, 42, 26, 36, 73, 10, 60] aims to break the online linear computation barrier by introducing an offline preprocessing phase. We categorize these protocols into two main groups: client-preprocessing PIR (CP-PIR) and server-preprocessing PIR (SP-PIR).

As detailed in [20, 76, 31, 42, 26, 36, 73, 10], CP-PIR protocols require the client to retrieve a private hint during the offline phase, which enables online queries to be evaluated with sublinear server-side computational cost in the database size. The client-preprocessing model for single-server PIR was first formalized by Corrigan-Gibbs and Kogan [20] and has since inspired a series of constructions [76, 31, 42, 26, 36, 73, 10]. While these protocols follow a similar framework in the online phase, the key distinctions lie in the procedure of private hint retrieval during the offline phase. In particular, two main approaches have been proposed for retrieving the private hint. One common approach is to stream the entire database from the server to the client, who then derives the hint locally [76, 42, 36, 73]. Another approach has the server evaluate the hint using FHE and return the encrypted result to the client [20, 75, 26].

Different from CP-PIR, SP-PIR (also known as doubly-efficient PIR) directly preprocesses and expands the database on the server side during the offline phase, thereby enabling sublinear online server computation without additional storage on the client. Existing SP-PIR protocols [47, 60] primarily focuses on reducing the computational cost of this server-side preprocessing.

Permutating encrypted data over FHE. Since our protocol relies on a specific form of homomorphic permutation, which we refer to as oblivious homomorphic rotation, we provide a review of existing homomorphic permutation techniques. Standard methods for rotating encrypted vectors [29] leverage the Galois automorphism to shift data across ciphertext slots. While highly efficient and supported by most FHE schemes, these methods require the rotation offset to be public. To support encrypted offsets, blind rotation was introduced as a core component of FHEW/TFHE bootstrapping [24, 19]. However, the blind rotation technique operates on coefficients in a negacyclic ring modulo X^n+1, which results in a negacyclic rotation. More general homomorphic permutation techniques have been extensively studied [16, 74, 26, 53], typically relying on permutation networks such as the Waksman network [71], the Thorp shuffle [70], or homomorphic matrix-vector multiplication. These works primarily focus on reducing the circuit size required for FHE evaluation. We observe that rotation is a structured subset of a general permutation with significantly lower circuit complexity and adopt rotation as the core primitive in RotPIR to reduce the computation overhead.

2 Preliminaries

We use bold lowercase letters (e.g., $\mathbf{a}$) for vectors, tilde lowercase letters (e.g., $\tilde{a}$) for polynomials, and bold uppercase letters (e.g., $\mathbf{A}$) for matrices. Throughout this work, we use λ to denote the security parameter, κ for correctness parameter, p for plaintext modulus, q for ciphertext modulus, and n to specify lattice dimension. $\mathbb{Z}_q$ refers to the set of integers modulo q. We define R and R_q denote $\mathbb{Z}[X]/(X^n+1)$ and $\mathbb{Z}[X]/(X^n+1) \bmod q$. For integer x, we let $[x]$ denote the set $\{0,\ldots,x-1\}$. The notation $\tilde{O}(f(n))$ hides poly-logarithmic factors in the parameter n, and $\tilde{O}_\lambda(\cdot)$ hides $\mathrm{poly}(\log n, \lambda)$ factors. PPT stands for *Probabilistic Polynomial Time*. For a comprehensive list of notations, please refer to the Section C.

2.1 Private Information Retrieval

We recall the formal definition of a single-server client-preprocessing PIR protocol adapted from [20, 76, 26].

Definition 2.1 *A single-server PIR protocol with client-preprocessing (CP-PIR)* Π *consists of six polynomial-time algorithms as follows.*

- $\mathsf{HintQuery}(1^\lambda, 1^\kappa, 1^N) \to (\mathsf{st}, \mathsf{qu}_0)$*, a randomized algorithm run by the client that takes as input the security parameter* λ*, the correctness parameter* κ*, and the database size* N*, and outputs an internal client state* st *and a* hint query qu_0*.*
- $\mathsf{HintAnswer}(\mathbf{D}, \mathsf{qu}_0) \to \mathsf{res}_0$*, a deterministic algorithm run by the server that takes a database* $\mathbf{D}$ *and the hint query* qu_0*, and outputs a* hint response res_0*.*
- $\mathsf{HintExtract}(\mathsf{st}, \mathsf{res}_0) \to \mathsf{hint}$*, a deterministic algorithm run by the client that takes the internal state* st *and the hint response* res_0 *as input, and outputs a* hint hint*, which is used in the online phase.*
- $\mathsf{Query}(\mathsf{st}, \mathsf{idx}) \to (\mathsf{st}', \mathsf{qu})$*, a randomized algorithm run by the client that takes the internal state* st *and an index* $\mathsf{idx} \in [N]$ *as input, and returns an updated state* st' *with an* online query qu*.*
- $\mathsf{Answer}(\mathbf{D}, \mathsf{qu}) \to \mathsf{res}$*, a deterministic algorithm run by the server that takes the database* $\mathbf{D}$ *and the online query* qu *as input, and returns the* online response res*.*
- $\mathsf{ReConstruct}(\mathsf{st}', \mathsf{hint}, \mathsf{res}) \to (\mathsf{hint}', \mathsf{val})$*, a deterministic algorithm run by the client that takes the state* st'*, the hint* hint*, and the online response* res *as input, and outputs the updated hint* hint' *and the result* $\mathsf{val} = \mathbf{D}[\mathsf{idx}]$*.*

We note that in the definition above, both the offline phase, consisting of the triple ($\mathsf{HintQuery}$, $\mathsf{HintAnswer}$, $\mathsf{HintExtract}$), and the online phase, consisting of (Query, Answer, $\mathsf{ReConstruct}$), are implemented by at most one message from client to server followed by a single message from server to client. This covers the vast majority of constructions [76, 36, 26, 20, 65, 75] of single-server PIR protocols with client-preprocessing.

Following [21], in Definition 2.2 we formally define the notion of correctness that we adopt in this paper and the notion of security against an honest-but-curious server is presented in Definition 2.3. This is the notion of security adopted by the vast majority of the work on PIR.

Definition 2.2 (Correctness.) *Let* $Q(\cdot)$ *be a polynomial. We say that a single-server CP-PIR protocol* Π *is* Q-query correct *if for every* $\lambda, \kappa \in \mathbb{N}$*, all databases* $\mathbf{D}$ *of size* N *polynomial in* κ*, and all* $(\mathsf{idx}_1, \ldots, \mathsf{idx}_{Q(\kappa)}) \in [N]^{Q(\kappa)}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that:*

$$\Pr\left[\mathsf{Corr}(\lambda, \kappa, \mathbf{D}, (\mathsf{idx}_1, \ldots, \mathsf{idx}_Q)) \to 1\right] \geq 1 - \mathsf{negl}(\kappa).$$

where the correctness game Corr *is defined as in Figure 2.*

Correctness Game $\mathsf{Corr}(\lambda, \kappa, \mathbf{D}, (\mathsf{idx}_1, \ldots, \mathsf{idx}_Q))$

- $(\mathsf{st}_0, \mathsf{qu}_0) \leftarrow \Pi.\mathsf{HintQuery}(1^\lambda, 1^\kappa, 1^{|\mathbf{D}|})$.
- $\mathsf{res}_0 \leftarrow \Pi.\mathsf{HintAnswer}(\mathbf{D}, \mathsf{qu}_0)$.
- $\mathsf{hint}_0 \leftarrow \Pi.\mathsf{HintExtract}(\mathsf{st}_0, \mathsf{res}_0)$.
- For $t = 1, \ldots, Q$:
  - $(\mathsf{st}_t, \mathsf{qu}_t) \leftarrow \Pi.\mathsf{Query}(\mathsf{st}_{t-1}, \mathsf{idx}_t)$
  - $\mathsf{res}_t \leftarrow \Pi.\mathsf{Answer}(\mathbf{D}, \mathsf{qu}_t)$
  - $(\mathsf{hint}_t, \mathsf{val}_t) \leftarrow \Pi.\mathsf{ReConstruct}(\mathsf{st}_t, \mathsf{hint}_{t-1}, \mathsf{res}_t)$
- Output 1 if $\mathsf{val}_t = \mathbf{D}[\mathsf{idx}_t]$, for all $t = 1, \ldots, Q$.
- Output 0 otherwise.

Figure 2: The correctness game for client-preprocessing PIR.

Security Game $\mathsf{Priv}^b_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q)$

Offline Phase

- $(\mathsf{st}_0, \mathsf{qu}_0) \leftarrow \Pi.\mathsf{HintQuery}(1^\lambda, 1^\kappa, 1^{|\mathbf{D}|})$.
- $\mathsf{res}_0 \leftarrow \Pi.\mathsf{HintAnswer}(\mathbf{D}, \mathsf{qu}_0)$.
- $\mathsf{Advst} \leftarrow \mathcal{A}(1^\lambda, \mathbf{D}, \mathsf{qu}_0, \mathsf{res}_0)$.

Online Phase

- For $i = 1, \ldots, Q$:
  - $(\mathsf{Advst}, \mathsf{idx}^0_i, \mathsf{idx}^1_i) \leftarrow \mathcal{A}(\mathsf{Advst})$
  - $(\mathsf{st}_t, \mathsf{qu}_i) \leftarrow \Pi.\mathsf{Query}(\mathsf{st}_{t-1}, \mathsf{idx}^b_i)$
  - $\mathsf{Advst} \leftarrow \mathcal{A}(\mathsf{Advst}, \mathsf{qu}_i)$
- Return $\mathcal{A}(\mathsf{Advst})$

Figure 3: The security game for client-preprocessing PIR.

Definition 2.3 (Security.) *Let $Q(\cdot)$ be a polynomial. We say that a single-server CP-PIR protocol Π is Q-*query secure*, if for every $\lambda, \kappa \in \mathbb{N}$, for all PPT adversaries $\mathcal{A}$, and for all databases $\mathbf{D}$ of size N polynomial in λ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\begin{aligned} |\ \Pr[\mathsf{Priv}^0_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q(\lambda)) \to 1] - \\ \Pr[\mathsf{Priv}^1_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q(\lambda)) \to 1]\ | \leq \mathsf{negl}(\lambda). \end{aligned}$$

where $\mathsf{Priv}^b_{\mathcal{A}}$ is defined as in Figure 3 and is parameterized by a PPT adversary $\mathcal{A}$ with a bit $b \in \{0, 1\}$, and takes as input the security parameter λ, the correctness parameter κ, the database $\mathbf{D}$, and the number Q of queries.

2.2 Homomorphic Encryption

We consider a symmetric-key fully homomorphic encryption (FHE) scheme with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. The scheme contains the following algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{evk})$: a randomized algorithm that takes the security parameter λ and outputs a secret

key sk and a set of public evaluation keys evk required for homomorphic operations.

- $\mathsf{Enc}(\mathsf{sk}, m) \rightarrow ct$: a randomized algorithm that takes as input a secret key sk and an abstract message $m \in \mathcal{M}$, and outputs a ciphertext $ct \in \mathcal{C}$ encrypting m.
- $\mathsf{Dec}(sk, ct) \rightarrow m$: a deterministic algorithm that takes as input a secret key sk and a ciphertext $ct \in \mathcal{C}$, and outputs the plaintext message $m \in \mathcal{M}$.

In this work, we focus on an FHE setting where the ciphertext space $\mathcal{C}$ consists of cross-scheme ciphertext formats, and supports the homomorphic operators defined below.

2.2.1 Ciphertext Types

In our concrete instantiation, the ciphertext space $\mathcal{C}$ is realized with two types of fundamental FHE ciphertexts: ring-learning-with-errors (RLWE) ciphertext RLWE and ring-Gentry-Sahai-Waters (GSW) ciphertext RGSW along with state-of-the-art optimization techniques [29, 25, 14, 68, 24, 19, 32, 2, 50].

• $\mathsf{RLWE}_{\tilde{s}}^{n,q}(\tilde{m})$: For RLWE ciphertext, we use the symmetric version of the BFV scheme [25], the encryption of a plain polynomial $\tilde{m} \in R_p$ under the secret $\tilde{s} \in R_q$ is:

$$\mathsf{RLWE}_{\tilde{s}}^{n,q}(\tilde{m}) = (\tilde{b}, \tilde{a}) = (-\tilde{a} \cdot \tilde{s} + \lceil q/p \rfloor \cdot \tilde{m} + \tilde{e}, \tilde{a}).$$

By setting the plain modulus $p \equiv 1 \mod 2n$, the SIMD technique [68] enables encoding a message vector $\mathbf{m} \in \mathbb{Z}_p^n$ into a polynomial $\tilde{m} \in R_p$. We refer to this as $\mathsf{RLWE}(\mathbf{m})$.

• $\mathsf{RGSW}_{\tilde{s}}^{n,q}(m)$: First introduced in [30] and later extended in [24, 19], given a gadget vector $\mathbf{g} = (g_0, g_1, ..., g_{L-1}) \in \mathbb{Z}_q^L$, the RGSW encryption of a plain polynomial $\tilde{m} \in R_p$ under the secret key $\tilde{s} \in R_q$ is:

$$\mathsf{RGSW}_{\tilde{s}}^{n,q}(\tilde{m}) = (\{\mathsf{RLWE}(g_i\tilde{m})\}_{i=0}^{L-1}, \{\mathsf{RLWE}(g_i\tilde{s} \cdot \tilde{m})\}_{i=0}^{L-1}).$$

Roughly speaking, we can consider an RGSW ciphertext as a set of $2L$ RLWE ciphertexts.

2.2.2 Homomorphic Operators

Here, we explain the key homomorphic operators in this work. We abbreviate the ciphertext notations to shorthands such as $\mathsf{RLWE}(\tilde{m}), \mathsf{RLWE}(\mathbf{m})$ and $\mathsf{RGSW}(\tilde{m})$ when the parameters are irrelevant from the discussion.

• $+$, $-$: We use standard ciphertext addition and subtraction operators over $\mathsf{RLWE}/\mathsf{RGSW}$ ciphertexts.

• $\odot$: Slot-wise multiplication under the slot-encoding RLWE ciphertext. Given two plaintext vectors $\mathbf{m}_0$ and $\mathbf{m}_1$, it holds that $\mathbf{m}_0 \odot \mathsf{RLWE}(\mathbf{m}_1) = \mathsf{RLWE}(\mathbf{m}_0 \odot \mathbf{m}_1)$ and $\mathsf{RLWE}(\mathbf{m}_0) \odot \mathsf{RLWE}(\mathbf{m}_1) = \mathsf{RLWE}(\mathbf{m}_0 \odot \mathbf{m}_1)$.

• $\boxdot$: $\mathsf{ExternalProduct}$ is a type of homomorphic multiplication where $\mathsf{RLWE}(\tilde{m}) \boxdot \mathsf{RGSW}(\tilde{d}) = \mathsf{RLWE}(\tilde{m} \cdot \tilde{d})$. In general, $\boxdot$ tends to be faster and generate less noise compared to a standard homomorphic multiplication [19].

• $\mathsf{HERotate}(\mathsf{RLWE}(\mathbf{m}), \delta, \mathsf{evk})$: Given a slot-encoding RLWE ciphertext $\mathsf{RLWE}(\mathbf{m})$, a rotation offset δ, and an evaluation key evk, the homomorphic rotation $\mathsf{HERotate}$ operator outputs a ciphertext $\mathsf{RLWE}(\mathbf{m} \ll \delta)$.

Remark: We note that the standard homomorphic rotation operator $\mathsf{HERotate}$ requires the rotation offset δ to be plaintext [29, 33], whereas in our setting it must remain hidden. This leads to the notion of oblivious homomorphic rotation, where the server performs rotations without learning the offset. Here, we define the oblivious homomorphic rotation operator $\mathsf{ObHERotate}$ and in Section 5, we present our concrete construction of this operator and how $\mathsf{ObHERotate}$ can be efficiently applied in RotPIR.

• $\mathsf{ObHERotate}(\mathbf{m}, ct_\delta, \mathsf{evk})$: Given a plaintext vector $\mathbf{m} = (m_0, m_1, \ldots, m_{n-1})$, a set of ciphertext ct_δ encrypting a rotation offset $\delta \in [n]$, and an evaluation key evk, the oblivious homomorphic rotation operator $\mathsf{ObHERotate}$ outputs a ciphertext encrypting $\mathbf{m}_\delta = (m_\delta, m_{\delta+1}, \ldots, m_{n-1}, m_0, \ldots, m_{\delta-1})$ without revealing δ.

3 The RotPIR Protocol

In this section, we first present the RotPIR protocol for the single-query case ($Q = 1$) in Section 3.1, and then analyze correctness and security in Section 3.2. In the next section, we extend the protocol to support multiple queries.

3.1 Protocol Overview

We represent the N-element database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix $\mathbf{D}$ and $\mathbf{D}_i$ denotes the i-th row of the database. Accordingly, each index $\mathsf{idx} \in [N]$ is associated with a pair $(u, v) \in [\sqrt{N}]^2$, where u is the row index and v is the column index of idx. In what follows, we outline the main procedures of RotPIR and illustrate the protocol in Figure 4.

Offline phase. In the offline phase, the client and server interact once, after which the client obtains hints that will be used to perform the online query efficiently.

1. The client generates a random secret key sk and the evaluation key evk for FHE, together with $\sqrt{N}$ random rotation offsets $\boldsymbol{\delta} = \{\delta_i\}_{i=0}^{\sqrt{N}-1}, \delta_i \in [\sqrt{N}]$ and one random rotation offset $r \in [\sqrt{N}]$. Subsequently, the $\sqrt{N}+1$ offsets are encrypted under sk, resulting in ciphertexts $\mathtt{ct}_0, \mathtt{ct}_1, \ldots, \mathtt{ct}_{\sqrt{N}-1}, \mathtt{ct}_r$. After that, the client sends the hint query qu consisting of the $\sqrt{N}+1$ ciphertexts and the evaluation key evk to the server.
2. Given the hint query $\mathsf{qu} = (\mathtt{ct}_0, \ldots, \mathtt{ct}_{\sqrt{N}-1}, \mathtt{ct}_r, \mathsf{evk})$, the server homomorphically computes two types of offline hints. First, for each row $\mathbf{D}_i$, it evaluates

$$\mathtt{cr}_i \leftarrow \mathsf{ObHERotate}(\mathbf{D}_i, \mathtt{ct}_i, \mathsf{evk})$$

   where $\mathsf{ObHERotate}$ denotes the oblivious homomorphic rotation defined in Section 2.2.2 which circularly rotates the vector $\mathbf{D}_i$ to the left by δ_i without revealing the offset. More details about the $\mathsf{ObHERotate}$ are presented in Section 5. After the rotations, the server obtains $\sqrt{N}$ ciphertexts $\mathtt{cr}_0, \mathtt{cr}_1, \ldots, \mathtt{cr}_{\sqrt{N}-1}$, where each $\mathtt{cr}_i$ encrypts the rotated vector $\mathbf{D}_i \ll \delta_i$. The server then homomorphically adds these ciphertexts as $\mathtt{ch} \leftarrow \mathtt{cr}_0 + \mathtt{cr}_1 + \cdots + \mathtt{cr}_{\sqrt{N}-1}$. The resulting ciphertext $\mathtt{ch}$ encrypts the aggregated vector $\sum_{i=0}^{\sqrt{N}-1}(\mathbf{D}_i \ll \delta_i)$. Next, using the encrypted offset $\mathtt{ct}_r$, the server extracts the same hidden position r from every row. This gives an encrypted ciphertext $\mathtt{cz}$ encrypting $\mathbf{z} = (\mathbf{D}_0[r], \ldots, \mathbf{D}_{\sqrt{N}-1}[r])$. The server returns the hint response $\mathsf{res}_0 = (\mathtt{ch}, \mathtt{cz})$ that is transmitted to the client.
3. At the end of the offline phase, the client decrypts the hint response res_0 to obtain the hint $\mathsf{hint} = (\mathbf{h}, \mathbf{z})$.

Online phase. In the online phase, the client takes as input an index $\mathsf{idx} = (u, v) \in [\sqrt{N}]^2$ and privately retrieves the corresponding item. At this phase, the client holds the secret key sk, the rotation offsets $\boldsymbol{\delta}$ and r, and the hint $\mathsf{hint} = (\mathbf{h}, \mathbf{z})$. The detailed process consists of three steps.

1. The client first computes the $\mathsf{ind} = v - \delta_u \pmod{\sqrt{N}}$. By construction, the hint entry $\mathbf{h}_{\mathsf{ind}}$ is an aggregated value that includes the target database element $\mathbf{D}[u, v]$. The client then takes the value r and construct the query $\mathsf{qu} = (s_0, \ldots, s_{\sqrt{N}-1})$ as follows:

$$s_i = \begin{cases} \mathsf{ind} + \delta_i \pmod{\sqrt{N}}, & \text{if } i \in [\sqrt{N}] \setminus \{u\}; \\ r, & \text{if } i = u. \end{cases}$$

   The client then sends the online query qu to the server.
2. After receiving the query $\mathsf{qu} = (s_0, \ldots, s_{\sqrt{N}-1})$ from the client, the server returns $\mathsf{res} = \bigoplus_{i=0}^{\sqrt{N}-1} \mathbf{D}_i[s_i]$.
3. Finally, the client reconstructs the desired database value val by $\mathbf{D}[u, v] = \mathbf{h}_{\mathsf{ind}} \oplus \mathbf{z}_u \oplus \mathsf{res}$.

We outline the protocol in Figure 4 and present its correctness and security in Theorem 3.1.

Construction: RotPIR for one query. We employ a homomorphic encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ObHERotate})$.

I. Offline phase.

$\underline{\mathsf{HintQuery}(1^\lambda, 1^N) \to (\mathsf{st}, \mathsf{qu}_0)}$.

// λ is the security parameter and N the size of the database.

1. $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
2. Sample $\sqrt{N}$ random rotations $\boldsymbol{\delta} = (\delta_0, \ldots, \delta_{\sqrt{N}-1}), \delta_i \in [\sqrt{N}]$ and one random rotation offset $r \in [\sqrt{N}]$.
3. Encrypt the $\sqrt{N}$ rotations: $\mathtt{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \delta_i), i \in [\sqrt{N}]$.
4. Encrypt the rotation offset: $\mathtt{ct}_r \leftarrow \mathsf{Enc}(\mathsf{sk}, r)$.
5. Output $\{\mathsf{st} = (\mathsf{sk}, \boldsymbol{\delta}, r), \mathsf{qu}_0 = (\mathtt{ct}_0, \ldots, \mathtt{ct}_{\sqrt{N}-1}, \mathtt{ct}_r, \mathsf{evk})\}$

$\underline{\mathsf{HintAnswer}(\mathbf{D}, \mathsf{qu}_0) \to \mathsf{res}_0}$.

1. Write qu_0 as $\mathsf{qu}_0 = (\mathtt{ct}_0, \ldots, \mathtt{ct}_{\sqrt{N}-1}, \mathtt{ct}_r, \mathsf{evk})$.
2. For $i \in [\sqrt{N}]$ do:

   $\mathtt{cr}_i \leftarrow \mathsf{ObHERotate}(\mathbf{D}_i, \mathtt{ct}_i, \mathsf{evk})$.

   $\mathtt{cz}_i \leftarrow \mathsf{ObHERotate}(\mathbf{D}_i, \mathtt{ct}_r, \mathsf{evk})$.

   // $\mathtt{cr}_i$ encrypts $\mathbf{D}_i \ll \delta_i$ and $\mathtt{cz}_i$ encrypts $\mathbf{D}_i \ll r$.
3. Compute $\mathtt{ch} \leftarrow \mathtt{cr}_0 + \ldots + \mathtt{cr}_{\sqrt{N}-1}$.
   // $\mathtt{ch}$ encrypts the vector $\sum_{i=0}^{\sqrt{N}-1}(\mathbf{D}_i \ll \delta_i)$.
4. Set $\mathtt{cz} \leftarrow \{\mathtt{cz}_0[0], \cdots, \mathtt{cz}_{\sqrt{N}-1}[0]\}$
   // $\mathtt{cz}$ encrypts the vector $\{\mathbf{D}_0[r], \cdots, \mathbf{D}_{\sqrt{N}-1}[r]\}$.
5. Output $\mathsf{res}_0 \leftarrow (\mathtt{ch}, \mathtt{cz})$.

$\underline{\mathsf{HintExtract}(\mathsf{st}, \mathsf{res}_0) \to \mathsf{hint}}$

1. Write st as $\mathsf{st} = (\mathsf{sk}, \boldsymbol{\delta}, r)$ and res_0 as $\mathsf{res}_0 = (\mathtt{ch}, \mathtt{cz})$.
2. Compute $\mathbf{h} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathtt{ch})$.
   // $\mathbf{h} = \sum_{i=0}^{\sqrt{N}-1}(\mathbf{D}_i \ll \delta_i)$.
3. Compute $\mathbf{z} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathtt{cz})$.
   // $\mathbf{z} = \{\mathbf{D}_0[r], \cdots, \mathbf{D}_{\sqrt{N}-1}[r]\}$.
4. Write $\mathsf{hint} = (\mathbf{h}, \mathbf{z})$ and return hint.

II. Online phase.

$\underline{\mathsf{Query}(\mathsf{st}, \mathsf{idx}) \to (\mathsf{st}', \mathsf{qu})}$

1. Write st as $\mathsf{st} = (\mathsf{sk}, \boldsymbol{\delta}, r)$.
2. Find the unique $u, v \in [\sqrt{N}]$ such that $\mathsf{idx} = u \cdot \sqrt{N} + v$.
3. Set $\mathsf{ind} = v - \delta_u \bmod \sqrt{N}$.
4. For $i \in [\sqrt{N}] \setminus \{u\}$, set $s_i = \mathsf{ind} + \delta_i \pmod{\sqrt{N}}$.
5. Set s_u equal to the value r.
6. Return $\mathsf{st}' = (\mathsf{sk}, u, \mathsf{ind})$, $\mathsf{qu} = (s_0, \ldots, s_{\sqrt{N}-1})$.

$\underline{\mathsf{Answer}(\mathsf{qu}, \mathbf{D}) \to \mathsf{res}}$

1. Write qu as $\mathsf{qu} = (s_0, \ldots, s_{\sqrt{N}-1})$.
2. Return response $\mathsf{res} = \mathbf{D}_0[s_0] \oplus \ldots \oplus \mathbf{D}_{\sqrt{N}-1}[s_{\sqrt{N}-1}]$.

$\underline{\mathsf{ReConstruct}(\mathsf{st}', \mathsf{hint}, \mathsf{res}) \to (\mathsf{hint}', a)}$

1. Write st as $\mathsf{st} = (\mathsf{sk}, u, \mathsf{ind})$.
2. Return $\mathsf{hint}' = \emptyset$ and $a = \mathbf{h}_{\mathsf{ind}} \oplus \mathbf{z}_u \oplus \mathsf{res}$.

Figure 4: The single query protocol RotPIR. For 1 query, we set $\kappa := \lambda$ and omit κ from the description.

Shift Game $\mathsf{SGame}^0_{\mathcal{A}}(L, K)$

- Adversary $\mathcal{A}$ outputs $\mathsf{idx} = (u, v) \in ([L] \times [K])$.
- Sample L random values $\boldsymbol{\delta} = (\delta_0, \ldots, \delta_{L-1}) \in [K]^L$ and an independent random value $r \leftarrow [K]$.
- Compute $\mathsf{ind} = v - \delta_u \mod K$.
- For $i \in [L] \setminus \{u\}$, set $s_i = \mathsf{ind} + \delta_i \pmod K$.
- Set $s_u = r$.
- Set $S^0 = (s_0, \ldots, s_{L-1})$.
- Outputs $\mathcal{A}(S^0)$.

Shift Game $\mathsf{SGame}^1_{\mathcal{A}}(L, K)$

- Adversary $\mathcal{A}$ outputs $\mathsf{idx} = (u, v) \in ([L] \times [K])$.
- Randomly select L random values $r_0, \ldots, r_{L-1} \leftarrow [K]$.
- Set $S^1 = (r_0, \ldots, r_{L-1})$.
- Outputs $\mathcal{A}(S^1)$

Figure 5: Shift games $\mathsf{SGame}^0_{\mathcal{A}}(L, K)$ and $\mathsf{SGame}^1_{\mathcal{A}}(L, K)$.

Theorem 3.1 *Assume that the underlying homomorphic encryption scheme is semantically secure, the single-server client-preprocessing PIR protocol defined in Figure 4 is correct (Definition 2.2) and secure (Definition 2.3) for a single query. Furthermore, for a database of size N and security parameter λ, in the offline phase:*

- *the server runs in time $\tilde{O}_\lambda(N)$,*
- *the total communication is $\tilde{O}_\lambda(\sqrt{N})$.*

and in the online phase:

- *the server runs in time $O(\sqrt{N})$,*
- *the query communication is $O(\sqrt{N})$,*
- *the response communication is $O(1)$.*

3.2 Correctness and Security

We now establish the correctness and security of the RotPIR defined in Figure 4, thereby proving Theorem 3.1.

We first consider the correctness of RotPIR (see Figure 4). At a high level, the offline phase produces two hints. The first hint $\mathbf{h}$ is an aggregated vector, where each entry is obtained by selecting one database record from each row after a hidden row-wise rotation. The second hint $\mathbf{z}$ contains the entries $(\mathbf{D}_0[r], \ldots, \mathbf{D}_{\sqrt{N}-1}[r])$ at a hidden random column r. In the online phase, for a query index $\mathsf{idx} = (u, v)$, the client identifies the hint entry $\mathbf{h}_{\mathsf{ind}}$ that includes the target record $\mathbf{D}_u[v]$. Then, the client asks the server for a compressed response that aggregates the same records contributing to $\mathbf{h}_{\mathsf{ind}}$, except that the contribution from the target row is replaced by $\mathbf{D}_u[r]$. Since the client has also precomputed $\mathbf{z}_u = \mathbf{D}_u[r]$ in the offline phase, it can cancel this explicitly retrieves the database records as $\mathbf{D}_u[v] = \mathbf{h}_{\mathsf{ind}} \oplus \mathbf{z}_u \oplus \mathsf{res}$. Due to space constraints, we provide the formal correctness proof of RotPIR in Section A.1.

We now prove that the protocol in Figure 4 satisfies the security. The proof proceeds in two steps. First, we introduce two shift games, SGame^0 and SGame^1, and establish Lemma 3.1, which indicates that for any

$\mathsf{Experiment}^b_{\mathcal{A}}(\lambda, \mathbf{D})$.

- Generate the secret key sk and the evaluation key evk for security parameter λ.
- Sample L random values $\boldsymbol{\delta} = (\delta_0, \ldots, \delta_{L-1})$ from $[K]$ and an independent random $r \leftarrow [K]$..
- Encrypt the vector $\boldsymbol{\delta}$ and r and thus obtaining $\mathsf{qu}_0 = (\mathsf{Enc}(\mathsf{sk}, \delta_0), \ldots, \mathsf{Enc}(\mathsf{sk}, \delta_{L-1}, \mathsf{Enc}(\mathsf{sk}, r)))$.
- $\mathcal{A}$ sees the evaluation key evk and ciphertexts in qu_0, outputs
  - $\mathsf{idx}_0 = (u_0, v_0)$ and $\mathsf{idx}_1 = (u_1, v_1)$
- Set $\mathsf{ind} = v_b - \delta_{u_b} \pmod K$.
- For $i \in [L] \setminus \{u_b\}$, set $s^b_i = \mathsf{ind} + \delta_i$.
- Set $s^b_{u_b} = r$.
- Set $S^b = (s^b_0, \ldots, s^b_{L-1})$ and output $\mathcal{A}(S^b)$.

Figure 6: The $\mathsf{Experiment}^b_{\mathcal{A}}(\lambda, \mathbf{D})$.

fixed index (u, v), the query vector generated by the client is uniformly distributed even when the target row is filled with the offline random offset r. Based on Lemma 3.1, we construct $\mathsf{Experiment}^b_{\mathcal{A}}(\lambda, \mathbf{D})$ in Figure 6 and prove $\mathsf{Experiment}^0_{\mathcal{A}}(\lambda, \mathbf{D})$ and $\mathsf{Experiment}^1_{\mathcal{A}}(\lambda, \mathbf{D})$ are computationally indistinguishable, which implies that Figure 4 satisfies the security in Definition 2.3.

Shift Game. We present two games, SGame^0 and SGame^1, in Figure 5, which are parameterized by L and K for a general $L \times K$ database layout. In RotPIR, the parameters are instantiated as $L = K = \sqrt{N}$. For any $L, K \in \mathbb{N}$, SGame^0 samples a vector in $[K]^L$ according to the same rule as the Query algorithm in Figure 4 under a fixed index (u, v), whereas SGame^1 outputs a uniformly random vector in $[K]^L$. The indistinguishability of two games is captured by the following Lemma 3.1. Due to space constraints, we provide the proof of Lemma 3.1 in Section A.2.

Lemma 3.1 *For any adversary $\mathcal{A}$, for any $L, K \in \mathbb{N}$,*

$$\Pr[\mathsf{SGame}^0_{\mathcal{A}}(L, K) \to 1] = \Pr[\mathsf{SGame}^1_{\mathcal{A}}(L, K) \to 1].$$

Security Game. We now turn to the security analysis of the RotPIR protocol. We first construct $\mathsf{Experiment}^b_{\mathcal{A}}(\lambda, \mathbf{D})$ defined in Figure 6. The experiment is a specialization of the general CP-PIR security game in Definition 2.3 to the single-query setting, with the protocol instantiated as Figure 4. We then show that no efficient PPT adversary can distinguish between the cases $b = 0$ and $b = 1$ in the experiment, which is the following Lemma 3.2. Due to space constraints, the proof of Lemma 3.2 is deferred to Section A.2.

Lemma 3.2 *If the encryption scheme is IND-CPA secure, then the experiment $\mathsf{Experiment}^0_{\mathcal{A}}(\lambda, \mathbf{D})$ and experiment $\mathsf{Experiment}^1_{\mathcal{A}}(\lambda, \mathbf{D})$ are computationally indistinguishable. Consequently, the single-server client-preprocessing PIR protocol in Figure 4 is secure.*

4 Handling Multiple Queries

In this section, we extend the single-query construction of RotPIR to $1 < Q < N$ queries and present *mRotPIR*.

Our multi-query construction extends the single-query protocol and adopts a chunk-based database partitioning strategy inspired by [20]. Specifically, the database is pseudo-randomly partitioned into Q disjoint

chunks, each of size $C := N/Q$. Such a partition allows each chunk to be treated as an independent database for preprocessing. For a target correctness parameter κ, we choose the number ℓ of hints per chunk ensure that every sequence of Q queries can be successfully executed except with negligible probability $\mathsf{negl}(\kappa)$. Beyond the protocol design, we provide a tighter correctness bound that allows a smaller ℓ for the same guarantee, reducing computation and communication overhead. We present the formal protocol in Section 4.1 and analyze correctness and security in Section 4.2.

4.1 Protocol Overview

In Figure 7, we provide the formal construction of mRotPIR. The protocol ensures that any PPT adversary achieves at most $\mathsf{negl}(\lambda)$ advantage in the security game. Simultaneously, the probability that a query fails to return the correct result is bounded by $\mathsf{negl}(\kappa)$[1]. In what follows, we outline the main procedures.

Offline Phase. During the offline phase, the database D of size N is pseudo-randomly partitioned into Q disjoint chunks. As specified in Figure 7, each chunk is logically represented as an $M \times M$ matrix, where $M := \sqrt{N/Q}$. To support Q queries, the client generates ℓ precomputed hints for each chunk. The generation of each hint follows the same procedure as the single-query offline phase in Figure 4. Distinct from the single-query setting, the multi-query protocol maintains a persistent client-side cache `cache` that stores previously retrieved $(\mathsf{idx}, \mathsf{val})$ pairs to efficiently support repeated queries. Such a caching technique is commonly employed in single-server CP-PIR protocols [76, 26].

Online Phase. We now describe the online phase for a *query* index idx to retrieve the database value $\mathbf{D}[\mathsf{idx}]$. A central feature of the multi-query protocol is that the client executes the protocol on a *target* tidx, which is not necessarily equal to the *query* index idx. The client begins by determining the target index tidx. In the typical case, when idx has not been queried before, and the corresponding chunk has unused hints, the client sets $\mathsf{tidx} := \mathsf{idx}$. Otherwise, the client selects a fresh random index $\mathsf{tidx} \neq \mathsf{idx}$ that has not been queried before and whose chunk has at least one unused hint. If idx has been queried, the client can return the cached value for idx and execute the protocol on tidx. Alternatively, if the chunk containing idx has consumed all available hints, the client executes the protocol on another tidx, and the query for idx returns $\perp$. As we indicate in Lemma B.1, the second case occurs with negligible probability.

After determining the target index tidx, the client generates the query for tidx. The client first identifies the chunk c containing tidx and selects an unused hint associated with chunk c. By construction, such a hint always exists. The query for tidx is then constructed according to the single-query protocol in Figure 4 and sent to the server. Upon receiving the query, the server evaluates the query on each database chunk, following the same procedure in Figure 4, and returns the resulting Q responses to the client. After receiving the Q responses, the client chooses only the one relative to chunk c together with the corresponding hint to extract $\mathsf{tval} = \mathbf{D}[\mathsf{tidx}]$. The pair $(\mathsf{tidx}, \mathsf{tval})$ is then stored in the client-side `cache`, the selected hint is marked as used for chunk c, and the remaining $Q - 1$ responses are discarded.

Finally, the client looks for an entry $(\mathsf{idx}, \mathsf{val})$ in the client cache. If such an entry is found, val is returned, otherwise $\perp$ is returned.

4.2 Correctness and Security

Following Figure 7, we formally present the correctness, security of mRotPIR in the following Theorem 4.1. Due to the space limitation, we provide the formal proof of Theorem 4.1 in Section B.

Theorem 4.1 (Single-server multi-query CP-PIR) *mRotPIR presented in Figure 7 is a single-server multi-query CP-PIR protocol that is secure (Definition 2.3) and correct (Definition 2.2) for Q queries. Furthermore, in the offline phase:*

- *the server runs in time $\tilde{O}_{\lambda,\kappa}(N)$,*

[1] We note that, in Figure 7, we set the number of chunks to be equal to the number Q of queries, and for each chunk, we compute ℓ hints, which suffice to guarantee that the probability of error is $\mathsf{negl}(\kappa)$.

Construction: mRotPIR for Q query. The protocol is parameterized by the number of supported queries Q, a homomorphic encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ObHERotate})$, and a pseudorandom permutation $\mathsf{PRP} : \{0,1\}^\lambda \times [N] \to [N]$.

I. Offline phase.

$\underline{\mathsf{HintQuery}_Q(1^\lambda, 1^\kappa, 1^N) \to (\mathsf{st}, \mathsf{qu}_0)}$.

//λ, κ security and correctness parameter, N database size.

1. Sample PRP key $k_\pi \leftarrow \{0,1\}^\lambda$, $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
2. Set number of chunks $C := N/Q$ and set number of hints per chunk ℓ according to Lemma B.2 for error probability $\leq 2^{-\kappa}$. Set $M := \sqrt{N/Q}$ and store the values as $\mathsf{params} := (N, C, \ell, M)$
3. Set $\mathsf{st} = (\mathsf{params}, k_\pi, \mathsf{sk})$ and $\mathsf{qu}_0 = (k_\pi, \mathsf{evk})$.
4. For $q \in [Q]$ and $k \in [\ell]$ do:
   - Sample M random rotations $\boldsymbol{\delta}^{q,k} = (\delta_0^{q,k}, \ldots, \delta_{M-1}^{q,k})$ from $[M]$ and one random rotation $r^{q,k} \in [M]$.
   - Encrypt the rotations by computing $\{\mathtt{ct}_m^{q,k} = \mathsf{Enc}(\mathsf{sk}, \delta_m^{q,k})\}_{m=0}^{M-1}$ and $ct_r^{q,k} = \mathsf{Enc}(\mathsf{sk}, r^{q,k})$.
5. Set $\mathtt{cache} \leftarrow \emptyset$, and $\mathsf{st} = (\mathsf{params}, k_\pi, \mathsf{sk}, \{\boldsymbol{\delta}^{q,k}\}_{q\in[Q],k\in[\ell]}, \{r^{q,k}\}_{q\in[Q],k\in[\ell]}, \mathtt{cache})$.
6. Set $\mathsf{qu}_0^{q,k} = (\mathtt{ct}_0^{q,k}, \ldots, \mathtt{ct}_{M-1}^{q,k}, \mathtt{ct}_r^{q,k})_{q\in[Q],k\in[\ell]}$.
7. Set $\mathsf{qu}_0 = (k_\pi, \mathsf{evk}, \mathsf{qu}_0^{q,k})$ and return $(\mathsf{st}, \mathsf{qu}_0)$.

$\underline{\mathsf{HintAnswer}_Q(\mathbf{D}, \mathsf{qu}_0) \to \mathsf{res}_0}$.

1. Parse $\mathsf{qu}_0 = (k_\pi, \mathsf{evk}, \{\mathsf{qu}_0^{q,k}\}_{q\in[Q],k\in[\ell]})$.
2. *// Permute the database according to* $\mathsf{PRP}(k_\pi, \cdot)$.
   For $q \in [Q]$, the q-th chunk C^q consist of N/Q entries:
   $$C^q \leftarrow [\mathbf{D}_{\mathsf{PRP}(k_\pi, q\cdot C)} || \ldots || \mathbf{D}_{\mathsf{PRP}(k_\pi, q\cdot C+(C-1))}]$$
   that are arranged in a $M \times M$ matrix.
3. For $q \in [Q]$ and $k \in [\ell]$:
   - Write $\mathsf{qu}_0^{q,k}$ as $(\mathtt{ct}_0^{q,k}, \ldots, \mathtt{ct}_{M-1}^{q,k}, \mathtt{ct}_r^{q,k})$
   - For $m \in [M]$:
     - $\mathtt{cr}_m^{q,k} = \mathsf{ObHERotate}(C_m^q, \mathtt{ct}_m^{q,k}, \mathsf{evk})$
     - $\mathtt{cz}_m^{q,k} = \mathsf{ObHERotate}(C_m^q, \mathtt{ct}_r^{q,k}, \mathsf{evk})$.
   - $\mathtt{ch}^{q,k} = \mathtt{cr}_0^{q,k} + \ldots + \mathtt{cr}_{M-1}^{q,k}$.
   - $\mathtt{cz}^{q,k} = (\mathtt{cz}_0^{q,k}[0], \cdots, \mathtt{cz}_{M-1}^{q,k}[0])$.
4. Return $\mathsf{res}_0 = \{\mathsf{res}_0^{q,k} = (\mathtt{ch}^{q,k}, \mathtt{cz}^{q,k})\}_{q\in[Q],k\in[\ell]}$.

$\underline{\mathsf{HintExtract}(\mathsf{st}, \mathsf{res}_0) \to \mathsf{hint}}$.

1. Parse $\mathsf{st} = (\mathsf{params}, k_\pi, \mathsf{sk}, (\boldsymbol{\delta}^{q,k}, r^{q,k})_{q,k}, \mathtt{cache})$
2. Parse $\mathsf{res}_0^{q,k} = (\mathtt{ch}^{q,k}, \mathtt{cz}^{q,k})$.
3. for $q \in [Q]$ and $k \in [\ell]$ do:
   - Set $\mathbf{h}^{q,k} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathtt{ch}^{q,k}), \mathbf{z}^{q,k} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathtt{cz}^{q,k})$.
4. Return $\mathsf{hint} = \{\mathbf{h}^{q,k}, \mathbf{z}^{q,k}\}_{q\in[Q],k\in[\ell]}$.

II. Online phase.

$\underline{\mathsf{Query}_Q(\mathsf{st}, \mathsf{idx}) \to (\mathsf{st}', \mathsf{qu})}$.

1. Parse $\mathsf{st} = (\mathsf{params}, k_\pi, \mathsf{sk}, (\boldsymbol{\delta}^{q,k}, r^{q,k})_{q\in[Q],k\in[\ell]}, \mathtt{cache})$.
2. *// Determine the* target *index* tidx *and an unused* $\mathsf{hint}^{q,k}$ *for* tidx*'s chunk.*
   (a) If the pair $(\mathsf{idx}, _) \notin \mathtt{cache}$ then proceed to step (b); otherwise proceed to step (c).
   (b) Set $\mathsf{tidx} = \mathsf{idx}$, proceed to step (d).
   (c) Choose tidx at random from those not in cache. *//a repeated query is replaced with a fresh query*
   (d) Find the unique $q \in [Q]$ and $c \in [C]$ such that $\mathsf{PRP}(k_\pi, \mathsf{tidx}) = qC + c$. *//* tidx *is the c-th entry in chunk q*
   (e) Find unique $u, v \in [M]$ such that $c = u \cdot M + v$. *//* tidx *is in row u and column v of chunk q*
   (f) Find $k \in [\ell]$ be such that $\boldsymbol{\delta}^{q,k} \neq \perp$ and set $\mathsf{ind} = v - \delta_u^{q,k} \pmod M$. If no such k exists, then proceed to step (c).
   // If all hints for chunk q are used, we pick a random tidx
3. *// Preparing query for* tidx
   - For $i \in [M] \setminus \{u\}$, set $s_i = \mathsf{ind} + \delta_i^{q,k}$.
   - Set s_u equal to $r^{q,k}$.
   - Store $(q, k, u, \mathsf{ind}, \mathsf{idx}, \mathsf{tidx})$ in st.
   - Update the state to st' by setting $\boldsymbol{\delta}^{q,k} = \perp$.
   - Set query $\mathsf{qu} = (s_0, \ldots, s_{M-1})$ and return $(\mathsf{st}', \mathsf{qu})$

$\underline{\mathsf{Answer}(\mathbf{D}, \mathsf{qu}) \to \mathsf{res}}$

1. Parse query $\mathsf{qu} = (s_0, \ldots, s_{M-1})$.
2. For $r \in [Q]$ *// Apply the query to all Q chunks*
   Set $\mathsf{res}^r = \oplus_{m=0}^{M-1} \mathbf{D}[\mathsf{PRP}(k_\pi, r \cdot C + m \cdot M + s_m)]$.
3. Return $\mathsf{res} = (\mathsf{res}^0, \ldots, \mathsf{res}^{Q-1})$.

$\underline{\mathsf{ReConstruct}(\mathsf{st}', \mathsf{hint}, \mathsf{res}) \to (\mathsf{hint}', \mathsf{val})}$

1. Extract $q, k, u, \mathsf{ind}, \mathsf{idx}, \mathsf{tidx}, \mathtt{cache}$ from st' and extract $\mathsf{hint}^{q,k}$ from hint.
2. Parse $\mathsf{res} = (\mathsf{res}^0, \ldots, \mathsf{res}^{Q-1})$, $\mathsf{res}^q = (\mathsf{res}_0^q, \ldots, \mathsf{res}_{M-1}^q)$.
3. Set $x = \mathsf{hint}_{\mathsf{ind}}^{q,k} \oplus \mathbf{z}_u^{q,k} \oplus \mathsf{res}_m^q$ and add (tidx, x) to $\mathtt{cache}$.
4. Set $\mathsf{hint}^{q,k} = \perp$ and updates the local hint from hint to hint'.
5. Find a pair $(\mathsf{idx}, \mathbf{D}[\mathsf{idx}])$ in $\mathtt{cache}$ and $\mathsf{val} = \mathbf{D}[\mathsf{idx}]$. If no such pair is found, $\mathsf{val} = \perp$.
6. Return $(\mathsf{hint}', \mathsf{val})$.

Figure 7: The mRotPIR protocol for $1 < Q < N$ queries.

Algorithm 1: ObHERotate

Input : A data vector $\mathbf{m} = [m_0, m_1, ..., m_{n-1}]$.
Input : An RGSW vector encrypting the rotation δ: $\mathsf{C}_{\mathsf{RGSW}} = \{\mathsf{RGSW}(\delta_b)\}_{b=0}^{\log n-1}$ where $\delta_b \in \{0,1\}$ and $\delta = \sum_{b=0}^{\log n-1} \delta_b \cdot 2^b$.
Input : The homomorphic evaluation key evk.
Output: An RLWE ciphertext $ct = \mathsf{RLWE}(\mathbf{m_r})$ where $\mathbf{m_r} = [m_\delta, m_{\delta+1}, ..., m_{n-1}, m_0, ..., m_{\delta-1}]$.

$ct_0 \leftarrow (\mathbf{m}, 0)$; // $ct_0 = \mathsf{RLWE}(\mathbf{m})$
for $b = 0$ **to** $\log n - 1$ **do**
  $ct_r \leftarrow \mathsf{HERotate}(ct_b, 2^b, \mathsf{evk})$
  $ct_{b+1} \leftarrow (ct_r - ct_b) \boxdot \mathsf{RGSW}(\delta_b) + c_b$
Return: $ct_{\log n}$

- *the total communication is* $\tilde{O}_{\lambda,\kappa}(Q\sqrt{N/Q})$

and in the online phase:

- *the server runs in time* $O(Q \cdot \sqrt{N/Q})$ *per query,*
- *the query communication is* $O(\sqrt{N/Q})$ *per query,*
- *the response communication is* $O(Q)$ *per query.*

Tighter Correctness Bound. The correctness of mRotPIR relies on the probability that no chunk receives more than ℓ queries. The prior analysis of [20] already establishes that this event occurs only with negligible probability. We further explore this probability in Section B, obtaining a tighter bound that is useful for concrete parameter selection. Since offline communication, and the server preprocessing all scale linearly in ℓ, this directly reduces the concrete overhead of mRotPIR.

5 Oblivious Homomorphic Rotation

In this section, we delve into the methodology of efficient oblivious homomorphic rotation, a key primitive for constructing offline hints in RotPIR. We first introduce ObHERotate in Section 5.1 that leverages the bit-decomposed encrypted offset to rotate a data vector obliviously. To handle various vector lengths, we construct MulObHERotate and PackObHERotate in Section 5.2 to support multi-block and packed rotations, respectively. Finally, we adopt a novel ciphertext compression techniques to further reduce the communication in Section 5.3.

5.1 Encrypted Rotation

Why existing homomorphic rotations do not apply. As indicated in Figure 4, the HintAnswer algorithm in RotPIR requires the server to rotate each database row obliviously, i.e., without learning the rotation offset δ during the computation. However, we note that existing homomorphic rotation techniques [33, 19] may not be well-suited for RotPIR. On the one hand, the standard homomorphic rotation operator [33] in Section 2.2.2 necessitates the rotation offset in plaintext and cannot be directly applied in our setting. On the other hand, while blind rotation [19] techniques enables encrypted rotation, the rotation operates on coefficients in a negacyclic ring modulo $X^n + 1$, resulting in a negacyclic shift rather than the cyclic rotation required by RotPIR.

Oblivious homomorphic rotation operator ObHERotate. To address the above challenge, we devise a new oblivious homomorphic rotation operator ObHERotate in Algorithm 1 tailored for RotPIR. The key idea of ObHERotate is to represent the rotation offset δ in binary and provide its bits as RGSW-encrypted selectors. For the b-th bit, ObHERotate conditionally applies a 2^b-slot rotation using an encrypted selector bit. Specifically, as indicated in Algorithm 1, given a database row $\mathbf{m}$ stored on the server and $\log n$ RGSW

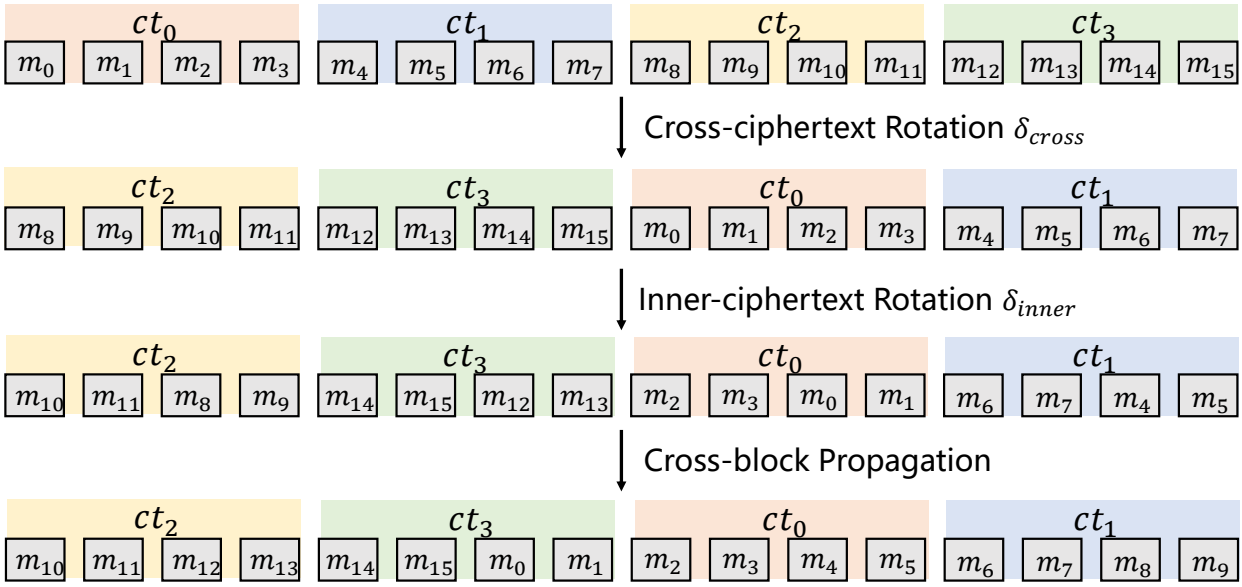


Figure 8: The conceptual illustrations of MulObHERotate with $N = 16, n = 4, K = 4, \delta = 10$.

ciphertexts provided by the client, the server first initializes the accumulator ciphertext as $ct_0 = (\mathbf{m}, 0)$. Next for each $b = 0, \ldots, \log n - 1$, the server computes a candidate ct_r by rotating ct_b with 2^b slots, and updates $ct_{b+1} \leftarrow (ct_r - ct_b) \boxdot \mathsf{RGSW}(\delta_b) + ct_b$, where $\boxdot$ denotes the external product defined in Section 2.2.2 and $\mathsf{RGSW}(\delta_b)$ is the RGSW encryption of the i-th bit of the rotation offset δ. If $\delta_b = 0$, the result ct_{b+1} is equal to ct_b, whereas if $\delta_b = 1$, the result ct_{b+1} is replaced with ct_r. After all iterations, the resulting ciphertext encrypts $\mathbf{m} \ll \delta$. Since both the rotation offset and the intermediate ciphertexts remain encrypted throughout the process, the server learns nothing about the rotation offset and rotation result. Due to the space limitation, we present the correctness and noise analysis of ObHERotate in Section D.

Limitations of ObHERotate. The operator ObHERotate performs an oblivious rotation over a vector encoded in a single RLWE ciphertext with ring dimension n (n SIMD slots). In practical PIR deployments, the vector length typically does not align with the SIMD capacity. For vectors longer than n, ObHERotate requires increasing the ring dimension n, which leads to significantly higher computation and communication costs. Conversely, for smaller vectors, ObHERotate still outputs an n-slot ciphertext, resulting in considerable communication overhead due to the unused SIMD slots, We note that this overhead cannot be avoided by simply reducing n, since practical homomorphic encryption schemes impose a security lower bound on the ring dimension n (e.g., $n \geq 1024$).

5.2 Multi-block and Packed Oblivious Rotations

To efficiently support oblivious rotation on both large and small vectors, we present two novel oblivious homomorphic rotation operators built upon ObHERotate, namely MulObHERotate for large vectors encoded in multiple ciphertexts and PackObHERotate for multiple independent short vectors packed into a single ciphertext.

Multi-block Oblivious Rotations. To handle vectors exceeding the SIMD capacity of a single RLWE ciphertext, we propose a multi-block oblivious homomorphic rotation operator MulObHERotate that can obliviously rotate the vector encoded in multiple ciphertexts. In this setting, we represent a large vector as a sequence of ciphertext blocks. The key idea is to decompose the global rotation offset δ into cross-ciphertext rotation offset δ_{cross} and inner-ciphertext rotation offset δ_{inner}. The cross-ciphertext rotation acts on the ciphertext sequence and obliviously shifts the order of ciphertexts, whereas the inner-ciphertext rotation

Algorithm 2: MulObHERotate

Input : A plain data vector $\mathbf{m} = [m_0, m_1, ..., m_{N-1}], N = K \cdot n$ where N is the vector length and n is the ring dimension.

Input : Two RGSW vectors encrypting the rotation δ: $\mathsf{C}^{\mathsf{cross}}_{\mathsf{RGSW}} = \{\mathsf{RGSW}(\delta_t^{\mathsf{cross}})\}_{t=0}^{\lceil \log K \rceil - 1}$ and $\mathsf{C}^{\mathsf{inner}}_{\mathsf{RGSW}} = \{\mathsf{RGSW}(\delta_s^{\mathsf{inner}})\}_{s=0}^{\lceil \log n \rceil - 1}$, where $\delta = \delta_{\mathsf{cross}} \cdot n + \delta_{\mathsf{inner}}$

Input : The homomorphic evaluation key evk.

Output: K RLWE ciphertext $ct = \{ct_0, ct_1, \cdots, ct_{K-1}\}$ where ct_i encrypts $(\mathbf{m} \ll \delta)_{[i \cdot n:(i+1) \cdot n]}$.

1 *// Block encoding: pack each length-n block into one ciphertext*
2 **for** $b = 0$ **to** $K - 1$ **do**
3 $\quad \mathbf{m}^b = [m_b, m_{b+1}, m_{b+2}, \cdots, m_{b+(n-1)}]$
4 $\quad ct_b \leftarrow (\mathbf{m}^b, 0)$; // $ct_b = \mathsf{RLWE}(\mathbf{m}^b)$
5 $ct \leftarrow \{ct_0, ct_1, \cdots, ct_{K-1}\}$
6 *// Cross-ciphertext rotation: apply the block shift by δ_{cross}.*
7 **for** $t = 0$ **to** $\lceil \log K \rceil - 1$ **do**
8 $\quad ct_{\mathsf{tmp}} \leftarrow ct$
9 $\quad$ **for** $b = 0$ **to** $K - 1$ **do**
10 $\quad\quad u \leftarrow ct_{\mathsf{tmp}}[b]$
11 $\quad\quad v \leftarrow ct_{\mathsf{tmp}}[b + 2^t \mod K]$
12 $\quad\quad ct[b] \leftarrow (v - u) \boxdot \mathsf{RGSW}(\delta_t^{\mathsf{cross}}) + u$
13 *// Inner-ciphertext rotation: shift within each ciphertext by δ_{inner} slots*
14 **for** $s = 0$ **to** $\lceil \log n \rceil - 1$ **do**
15 $\quad ct[b] \leftarrow \mathsf{ObHERotate}(ct[b], \mathsf{C}^{\mathsf{inner}}_{\mathsf{RGSW}}, \mathsf{evk})$
16 *// Contrust two selector ciphetexts: ct_{tail} and ct_{head}.*
17 *ct_{tail} encrypts a 0/1 vector where the last δ_{inner} slots are 1.*
18 *ct_{head} encrypts a 0/1 vector where the first $n - \delta_{\mathsf{inner}}$ slots are 1.*
19 $ct_{\mathsf{tail}} \leftarrow (0, 0)$; // $ct_{\mathsf{tail}} = \mathsf{RLWE}(0)$
20 **for** $s = 0$ **to** $\lceil \log n \rceil - 1$ **do**
21 $\quad \mathsf{Mask}[i] = \begin{cases} 0, & 0 \le i < 2^s, \\ 1, & n - 2^s \le i < n. \end{cases}$
22 $\quad u \leftarrow ct_{\mathsf{tail}}$; $v \leftarrow \mathsf{HERotate}(ct_{\mathsf{tail}}, 2^s, \mathsf{evk}) + \mathsf{Mask}$
23 $\quad ct_{\mathsf{tail}} \leftarrow (v - u) \boxdot \mathsf{RGSW}(\delta_s^{\mathsf{inner}}) + u$
24 $ct_{\mathsf{head}} \leftarrow [1, 1, \cdots, 1] - ct_{\mathsf{tail}}$
25 *// Cross-block propagation*
26 **for** $b = 0$ **to** $K - 1$ **do**
27 $\quad ct_b \leftarrow ct_b \odot ct_{\mathsf{head}} + ct_{(b+1) \mod K} \odot ct_{\mathsf{tail}}$

Return: $ct_0, ct_1, \cdots, ct_{K-1}$

rotates the SIMD slots within each ciphertext. In combination, the two oblivious rotations achieve a global oblivious rotation of the large data vector.

In Algorithm 2, we present the multi-block oblivious rotation operator MulObHERotate and take a 4-block ciphertext as an example to illustrate the procedure in Figure 8. Given as input a plaintext vector $\mathbf{m}$ of length $N = K \cdot n$ and RGSW ciphertexts encrypting the rotation offset δ, Algorithm 2 begins by encoding the plaintext vector into K ciphertext blocks $ct = \{ct_0, ct_1, \cdots, ct_{K-1}\}$ where ct_b encrypts the subvector $[m_b, m_{b+1}, m_{b+2}, \cdots, m_{b+(n-1)}]$. Subsequently, in Line 6-12, the algorithm performs the cross-ciphertext rotation according to the encrypted offset δ_{cross}. For each bit position t of δ_{cross}, the encrypted bit $\mathsf{RGSW}(\delta_b^{\mathsf{cross}})$ serves as a selector to obliviously decide whether to shift the ciphertext sequence by 2^t blocks. Iterating over all bits yields a cyclic oblivious rotation of the ciphertext sequence by δ_{cross}. Next, in Line 13-15, we apply the inner-ciphertext rotation to each ciphertext in ct by evaluating the ObHERotate operator, which obviously rotates the slots within each ciphertext by δ_{inner}.

After the cross-ciphertext and inner-ciphertext rotations, the last δ_{inner} slots of each result ciphertext still originate from the same block, whereas a correct global rotation requires these slots to come from the next ciphertext. Therefore, Lines 16–26 construct two encrypted selector masks ct_{tail} and ct_{head} where ct_{tail}

Algorithm 3: PackObHERotate

Input : K N-size plain data vectors $\{\mathbf{m}_0, \mathbf{m}_1, ..., \mathbf{m}_{K-1}\}, n = K \cdot N$.
Input : K RGSW vectors encrypting the rotations $\delta_0, \delta_1, \cdots, \delta_{K-1}$: $\mathsf{C}^k_{\mathsf{RGSW}} = \{\mathsf{RGSW}(\delta^k_t)\}^{\lceil \log N \rceil - 1}_{t=0}$ where $\delta^k = \sum_{t=0}^{\lceil \log N \rceil - 1} \delta^k_t \cdot 2^t$
Input : The homomorphic evaluation key evk.
Output: An RLWE ciphertext ct encrypts the concatenated vector $\mathbf{m_0} << \delta_0 \| \mathbf{m_1} << \delta_1 \| \cdots \| \mathbf{m_{K-1}} << \delta_{K-1}$.

1 *// Independent rotation*
2 **for** $k = 0$ **to** $K - 1$ **do**
3 $\quad \mathbf{m} \leftarrow \mathbf{m}_k \| \mathbf{m}_k \| \cdots \mathbf{m}_k \in \mathbb{Z}^n$
4 $\quad ct_k \leftarrow \mathsf{ObHERotate}(\mathbf{m}, \mathsf{C}^k_{\mathsf{RGSW}}, \mathsf{evk})$
5 *// Ciphertext packing*
6 $ct \leftarrow (0, 0)$ *// $ct = \mathsf{RLWE}(0)$*
7 **for** $k = 0$ **to** $K - 1$ **do**
8 $\quad \mathsf{Mask}^{(k)}[i] = \begin{cases} 1, & kN \le i < (k+1)N, \\ 0, & \text{otherwise}, \end{cases} \quad i = 0, 1, \ldots, n-1.$
9 $\quad ct_k \leftarrow \mathsf{Mask} \cdot ct_k$
10 $\quad ct \leftarrow ct + ct_k$
11 $\quad \mathsf{Mask} \leftarrow \mathsf{Mask} >> K$
Return: ct

Table 2: Comparison of homomorphic rotation techniques.

Scheme	Offset Hidden	Rotation Type	Multi-Block Vector^{+}	Packed Vector^{++}
Standard Rotation [33]	✗	Cyclic	✗	✗
Blind Rotation [19]	✓	Negacyclic	✗	✗
SHIP [18]	✓	Cyclic	✗	✗
RotPIR Rotation*	✓	Cyclic	✓	✓

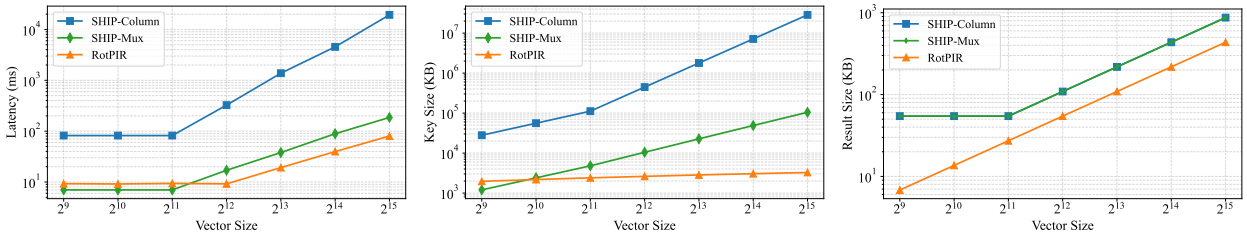
$^{+}$Supports vectors across multiple ciphertexts.
$^{++}$Supports packing multiple vectors in one ciphertext.

encrypts a 0/1 vector with ones in the last δ_{inner} slots and ct_{head} with ones in the first $n - \delta_{\mathsf{inner}}$ slots. Finally, in Line 27-29, we perform a cross-block propagation step by evaluating $ct_b \leftarrow ct_b \odot ct_{\mathsf{head}} + ct_{(b+1) \mod K} \odot ct_{\mathsf{tail}}$, which preserves the $n - \delta_{\mathsf{inner}}$ slots of ct_b and replaces the last δ_{inner} slots with ct_{b+1}.The final result ciphertexts encrypts the large rotated vector $\mathbf{m} \ll \delta$. Due to the space limitation, we provide the noise analysis of $\mathsf{MulObHERotate}$ in Section D.

Packed Oblivious Rotation. To handle the case where multiple short data vectors can be packed into a single RLWE ciphertext, we propose a packed oblivious homomorphic rotation operator $\mathsf{PackObHERotate}$. In this case, we rotate each vector obliviously with an independent offset and pack the result into the same ciphertext.

In Algorithm 3, we present the construction of the multi-block oblivious rotation operator $\mathsf{PackObHERotate}$. Given K plaintext vectors of length N and K independent RGSW ciphertext vectors encrypting the rotation offsets $\delta_0, \delta_1, \ldots, \delta_{K-1}$, Algorithm 3 first processes each vector $\mathbf{m}_k$ independently. In Line 1-4, for each k, the vector $\mathbf{m}_k$ is replicated K times to form the concatenated vector $\mathbf{m} \leftarrow \mathbf{m}_k \| \mathbf{m}_k \| \cdots \| \mathbf{m}_k$. The basic oblivious homomorphic rotation operator $\mathsf{ObHERotate}$ is then applied to $\mathbf{m}$ using the encrypted rotation offset δ_k,resulting in a ciphertext ct_k that encrypts $\mathbf{m}_k \ll \delta_k \| \mathbf{m}_k \ll \delta_k \| \cdots \| \mathbf{m}_k \ll \delta_k$. Next, in Line 6-11, the algorithm packs the rotated results into a single ciphertext. For each k, we construct a binary mask $\mathsf{Mask}^{(k)} \in \{0, 1\}^n$ that takes value one on indices $kN \le i < (k+1)N$ and zero elsewhere. Aftering evaluating slot-wise multiplication $\mathsf{Mask}^{(k)} \odot ct_k$, we extract the k-th segment of ct_k. Finally, summing the $\mathsf{Mask}^{(k)} \odot ct_k$ over all k yields the output ciphertext ct encrypting $\mathbf{m}_0 \ll \delta_0 \| \mathbf{m}_1 \ll \delta_1 \| \cdots \| \mathbf{m}_{K-1} \ll \delta_{K-1}$.

Remark: We note that the recent work SHIP [18] provides a similar primitive that homomorphically rotates



(a) Latency. (b) Size of the evaluation key. (c) Size of result ciphertexts.

Figure 9: Microbenchmark results of homomorphic oblivious rotation on a data vector size of N. We compare RotPIR against SHIP [18] on latency, size of the evaluation key, and size of result ciphertexts on a single CPU core.

a cleartext by an amount that is provided in encrypted form. While the method in [18] shares similarities with RotPIR, we achieve better efficiency than [18]. We provide a comparison of the homomorphic rotation technique in Table 2.

5.3 Query Compression

For an $\sqrt{N} \times \sqrt{N}$ databas, in ObHERotate, the client is required to send $\log \sqrt{N}$ RGSW ciphertext per database row, and amounts to transmitting $\sqrt{N} \log \sqrt{N}$ RGSW ciphertexts, which results in an excessively large communication overhead. To reduce the communication cost, we adopt the query compression technique to pack the hint query from $\sqrt{N} \log \sqrt{N}$ RGSW ciphertexts into RLWE ciphertexts and reconstruct the RGSW ciphertexts on the server side. We provide detail techniques in Section D.

6 Evaluation

Throughout the experiments, we wish to answer the following two main research questions (RQs).

- **RQ1**: How do the cryptographic components of RotPIR perform compared to SOTA methods?
- **RQ2**: How does the performace of mRotPIR compared to SOTA CP-PIR protocols in various databases?

We implemented RotPIR and mRotPIR in C++17, compiled with GCC 11.4.0, building upon Microsoft SEAL [66] for CPU-based homomorphic encryption operations and Troy [49] for GPU-accelerated implementations. The CPU experiments were conducted on an Intel(R) Xeon(R) Gold 5318Y processor with 512 GB of RAM, and the GPU experiments were conducted on an RTX 3090. We configured the parameters of RotPIR to provide at least $\lambda = 128$-bit of security level according to [3] and [4]. For homomorphic encryption, we set the $n = 4096$ and $p \approx 2^{16}, q \approx 2^{109}$ with 2 RNS moduli. The error distribution χ is chosen to be a discrete Gaussian with a standard deviation of $\sigma = 3.2$. For mRotPIR, we set $\ell = 17$, which yields a failure probability $< 2^{-40}$, the same as ThorPIR [26] and PIANO [76].

Handling Larger Records. For records larger than the plaintext space $p \approx 2^{16}$, we follow prior work [51, 35] and split each record into d smaller records, queried as d independent databases on which the same query is executed.

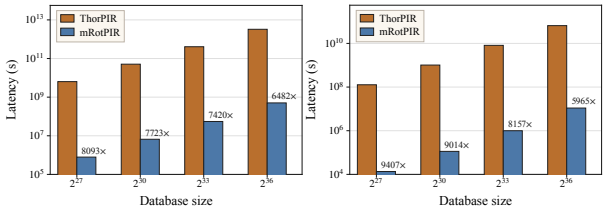
6.1 Microbenchmarks

To answer RQ1, we conducted various microbenchmarks to evaluate the efficiency of the cryptographic components in RotPIR, including oblivious rotation and hint generation.

Oblivious homomorphic rotation. We first benchmark the computation and communication cost of the oblivious homomorphic rotation under varying vector size N in Figure 9. We compare RotPIR with SHIP [18], a recent construction for oblivious homomorphic rotation, including both its column-based (SHIP-Column)

Table 3: Breakdown of offline hint generation in RotPIR.

Database		$N = 2^{16}$	$N = 2^{20}$	$N = 2^{24}$
CPU Time (sec)	QueryUnpack	3.82	15.01	65.26
	ObHERotate	1.56	8.09	39.78
	Total	5.38	23.10	105.04
GPU Time (sec)	QueryUnpack	0.06	0.24	1.16
	ObHERotate	0.03	0.13	0.80
	Total	0.09	0.37	1.96
Size (KB)	Hint Query	2737	2791	3282
	Hint Response	108	108	108
	Total	2845	2899	3390



(a) CPU computation. (b) GPU computation.

Figure 10: Offline server computation when varying the database size. We fix the record size to 360 bytes.

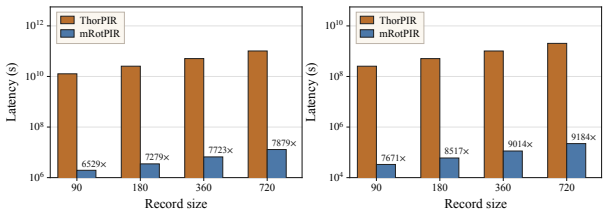
and multiplexer-based (SHIP-MUX) variants. We measure the latency, the size of the evaluation key, and the output size.

As indicated in Figure 9a, on large vectors ($N \geq 2^{12}$), RotPIR achieves 2×–241× lower latency than SHIP by leveraging the multi-block rotation operator MulObHERotate. For smaller vectors, RotPIR outperforms SHIP-Column by about 9× and performs comparably to SHIP-MUX. Regarding evaluation keys (Figure 9b), RotPIR reduces key size by up to 8738× compared to SHIP-Column and up to 32× compared to SHIP-MUX at large N, while SHIP-MUX is slightly smaller at small N. For output ciphertexts (Figure 9c), RotPIR achieves up to 8× smaller sizes at small N due to the packed rotation, and remains within a 2× factor as N increases.

Hint generation. We next evaluate the efficiency of the hint generation phase, which is the core offline component of RotPIR. Table 3 breaks down the runtime into two cryptographic operations: query unpacking QueryUnpack and oblivious rotation ObHERotate. On a single CPU, RotPIR generates hints for databases with $N = 2^{16}, 2^{20}, 2^{24}$ entries in 5.38s, 23.10s, and 105.04s, respectively. While in our GPU implementation, the runtime is reduced to $0.09s$, 0.37s, and 1.96s. The offline communication is also compact. The hint query is around 2.8–3.3 MB, while the hint response remains 108 KB across all tested database sizes.

6.2 PIR Benchmarks

To answer **RQ2**, we evaluate the offline computational and communication efficiency of mRotPIR across varying database sizes. We primarily compare against ThorPIR [26], the SOTA single-server CP-PIR protocol with sublinear offline communication, and PIANO [76], the SOTA single-server CP-PIR protocol with linear offline communication. For ThorPIR and mRotPIR, we set the query capacity to $Q = N^{1/3}$, matching the standard parameter choice of ThorPIR. Piano streams the entire database and therefore communication and server-side computation of the preprocessing phase are independent of the number of supported queries. For



(a) CPU computation. (b) GPU computation.

Figure 11: Offline server computation when varying the record size. We fix the database size to $N = 2^{30}$.

Database	Protocol	Offline						Online		Total Cost†
		Comm. (GB)	Comm. Cost	Ser. Comp. (h) 1 GPU	Ser. Comp. (h) 64 GPUs	Comp. Cost	Offline Cost	Comm. (KB)	Ser. Comp. (ms)	
$2^{30} \times 108$ B (108 GB)	ThorPIR	0.42	\$0.04	84934	1327	\$11041	\$11041	111	1.1	\$11041
	PIANO	108	\$9.72	0	0	\$0	\$9.72	121	7.51	\$9.76
	mRotPIR	26.9	\$2.42	10.6	0.17	\$0.95	**\$3.37**	109	12.97	**\$3.46**
$2^{30} \times 360$ B (360 GB)	ThorPIR	1.4	\$0.13	283115	4423.7	\$36805	\$36805	363	3.6	\$36805
	PIANO	360	\$32.40	0	0	\$0	\$32.40	121	13.72	\$32.49
	mRotPIR	83.4	\$7.51	31.4	0.5	\$4.08	**\$11.59**	361	30.95	**\$11.72**
$2^{33} \times 128$ B (1 TB)	ThorPIR	2.0	\$0.18	805306	12582.9	\$104690	\$104690	262	7.2	\$104690
	PIANO	1024	\$92.16	0	0	\$0	\$92.16	374	23.14	\$92.28
	mRotPIR	125.6	\$11.30	102.1	1.60	\$13.27	**\$24.57**	257	53.31	**\$24.85**

† Total cost = Offline cost + # supported queries Q × Online Cost, here we set $Q = N^{1/3}$.

Table 4: Estimated monetary cost of the offline and online server computation and communication across representative database sizes with $N = 2^{30}, 2^{33}$. We use \$0.13/GPU-hour for RTX 3090 rental on Akash Network [1] ,\$0.09/GB for data transfer, and $\$1.5 \times 10^{-5}$ per CPU core-second following AWS pricing [6].

ThorPIR [26], since no public implementation is available, we follow the concrete parameters and estimation methodology reported in the paper. For mRotPIR, we use the measured runtime of our CPU and GPU implementations and scale it over all chunks and independent hints used.

Figure 10 indicates the offline server computation as the database size grows, with the record size fixed to 360 bytes, while Figure 11 reports the corresponding results under varying record sizes with $N = 2^{30}$. As the database size increases, mRotPIR achieves 6482×–8093× speedup on CPU and 5965×–9407× speedup on GPU. As the record size increases, the speedup ranges from 6529× to 7879× on CPU and from 7671× to 9184× on GPU. These results demonstrate that mRotPIR substantially reduces the offline server computation compared to ThorPIR.

While mRotPIR achieves orders of magnitude improvement in offline server computation, it introduces a communication and computation trade-off. Table 4 summarizes the computation and communication trade-off and translates it into an illustrative monetary cost estimate across various database sizes. Compared with ThorPIR, mRotPIR uses higher offline communication. However, this additional communication leads to a substantial reduction in the dominant server-side FHE computation. For instance, on the 360 GB database, mRotPIR reduces the single-GPU offline computation from 283115 hours to 31.4 hours. Moreover, the latency could be reduced by more GPUs. With 64 GPUs, the projected latency drops to 0.5 hours on the 360 GB database and 1.60 hours on the 1 TB database. Compared with Piano, mRotPIR avoids downloading the whole database and reduce the communication cost from 360 GB and 1 TB to 83.4 GB and 125.6 GB, respectively.

To have a comprehensive quantitative estimation of communication and computation, Table 4 further estimates the monetary cost using two public pricing assumptions: GPU computation is charged at \$0.13/GPU-

hour for RTX 3090 rental on Akash Network [1], and data transfer is charged at \$0.09/GB following AWS pricing [6]. Under this cost model, mRotPIR achieves the lowest estimated monetary cost in all tested settings. On the 108 GB database, the preprocessing of mRotPIR costs \$3.37, which is 2.88× lower than PIANO and more than $>$ 1000× lower than ThorPIR. On the 360 GB database, mRotPIR costs \$11.59, compared with \$32.40 for PIANO and \$36805 for ThorPIR. On the 1 TB database, mRotPIR costs \$24.57, which is 3.75× lower than PIANO and $>$ 1000× lower than ThorPIR.

While our main contribution lies in reducing offline preprocessing overhead, We also report the online performance of mRotPIR compared to previous CP-PIR protocols. As indicated in Table 4, mRotPIR achieves online communication comparable to ThorPIR across all database sizes. The online server computation of mRotPIR is higher than that of ThorPIR and Piano, since the server evaluates the query over the chunked database structure. These results show that the offline savings of mRotPIR do not come at the cost of prohibitive online overhead. When the total monetary cost includes both offline preprocessing and the execution of Q online queries, mRotPIR remains the lowest-cost protocol in all tested settings. This shows that the offline savings of mRotPIR are not offset by its online query cost.

7 Conclusion

In this work, we introduced RotPIR, a single-server sublinear PIR protocol that can simultaneously achieves sublinear offline communication and computation-efficient offline preprocessing. We propose a novel protocol that leverages a novel hint construction strategy and devise an advanced oblivious homomorphic rotation operator to support efficient hint evaluation in RotPIR. We demonstrate that RotPIR can outperform the best-known sublinear PIR protocols on offline preprocessing. Despite the progress made by RotPIR, there is a need for latency improvement to adapt to real-world scenarios. One important future research direction would be to improve efficiency through more advanced cryptographic primitives or hardware-based accelerations.

References

[1] Akash Network. GPU pricing and availability. `https://akash.network/pricing/gpus/`, 2026. Accessed: June 2026.

[2] Ahmad Al Badawi, Yuriy Polyakov, Khin Mi Mi Aung, Bharadwaj Veeravalli, and Kurt Rohloff. Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme. *IEEE TETC*, 9(2):941–956, 2019.

[3] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. Homomorphic encryption standard. In *Protecting privacy through homomorphic encryption*, pages 31–62. Springer, 2022.

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

[5] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication–Computation trade-offs in PIR. In *USENIX Security Symposium*, pages 1811–1828, 2021.

[6] Amazon Web Services. Amazon EC2 on-demand pricing. `https://aws.amazon.com/ec2/pricing/on-demand/`, 2026. Accessed: June 2026.

[7] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *IEEE symposium on security and privacy*, pages 962–979, 2018.

[8] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 551–569, 2016.

[9] Apple Inc. Getting up-to-date calling and blocking information for your app. `https://developer.apple.com/documentation/identitylookup/getting-up-to-date-calling-and-blocking-information-for-your-app`, 2024. Accessed: Jan. 2026.

[10] Pranav Shriram Arunachalaramanan and Ling Ren. Single-server stateful PIR with verifiability and balanced efficiency. *Cryptology ePrint Archive*, 2025.

[11] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In *CRYPTO*, pages 55–73. Springer, 2000.

[12] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In *Theory of Cryptography Conference*, pages 662–693. Springer, 2017.

[13] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO*, pages 868–886, 2012.

[14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *TOCT*, 6(3):1–36, 2014.

[15] Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In *Information Security and Privacy*, pages 50–61, 2004.

[16] Hao Chen, Ilaria Chillotti, and Ling 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*, pages 345–360, 2019.

[17] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) LWE ciphertexts. In *International conference on applied cryptography and network security*, pages 460–479. Springer, 2021.

[18] Jung Hee Cheon, Guillaume Hanrot, Jongmin Kim, and Damien Stehlé. Ship: A shallow and highly parallelizable ckks bootstrapping algorithm. In *EUROCRYPT*, pages 398–428. Springer, 2025.

[19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[20] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In *EUROCRYPT*, pages 3–33. Springer, 2022.

[21] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *EUROCRYPT*, pages 44–75. Springer, 2020.

[22] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography*, pages 119–136, 2001.

[23] Alex Davidson, Gonçalo Pestana, and Sofía Celi. Frodopir: Simple, scalable, single-server private information retrieval. *Privacy Enhancing Technologies Symposium*, 2023.

[24] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015.

[25] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[26] Ben Fisch, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. ThorPIR: single server PIR via homomorphic Thorp shuffles. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1448–1462, 2024.

[27] Craig Gentry and Shai Halevi. Compressible fhe with applications to PIR. In *Theory of cryptography conference*, pages 438–464, 2019.

[28] Craig Gentry, Shai Halevi, and Nigel P Smart. Fully homomorphic encryption with polylog overhead. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 465–482. Springer, 2012.

[29] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*, pages 850–867. Springer, 2012.

[30] Craig Gentry, Amit Sahai, and Brent 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*, pages 75–92. Springer, 2013.

[31] Ashrujit Ghoshal, Mingxun Zhou, and Elaine Shi. Efficient pre-processing PIR without public-key cryptography. In *EUROCRYPT*, pages 210–240. Springer, 2024.

[32] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved RNS variant of the BFV homomorphic encryption scheme. In *The Cryptographers' Track at RSA Conference*, pages 83–105, 2019.

[33] Shai Halevi and Victor Shoup. Algorithms in helib. In *CRYPTO*, pages 554–571. Springer, 2014.

[34] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with Tiptoe. In *Symposium on Operating Systems Principles*, pages 396–416, 2023.

[35] Alexandra Henzinger, Matthew M Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast Single-Server private information retrieval. In *USENIX Security Symposium*, pages 3889–3905, 2023.

[36] Alexander Hoover, Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Plinko: single-server PIR with efficient updates via invertible PRFs. In *EUROCRYPT 2025*, pages 3–33.

[37] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In *USENIX security symposium*, pages 875–892, 2021.

[38] 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*, pages 364–373, 1997.

[39] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Annual IEEE Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[40] Albert Hyukjae Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. 2015.

[41] Arthur Lazzaretti and Charalampos Papamanthou. Near-optimal private information retrieval with preprocessing. In *Theory of Cryptography Conference*, pages 406–435. Springer, 2023.

[42] Arthur Lazzaretti and Charalampos Papamanthou. Single pass client-preprocessing private information retrieval. In *USENIX Security Symposium*, pages 5967–5984, 2024.

[43] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 227–256. Springer, 2023.

[44] Ryan Lehmkuhl, Alexandra Henzinger, and Henry Corrigan-Gibbs. Distributional private information retrieval. *Cryptology ePrint Archive*, 2025.

[45] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz-Wu. Hintless single-server private information retrieval. In *CRYPTO*, pages 183–217, 2024.

[46] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. Protocols for checking compromised credentials. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1387–1403, 2019.

[47] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic ram computation from ring lwe. In *ACM Symposium on Theory of Computing*, pages 595–608, 2023.

[48] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *Information Security*, pages 314–328, 2005.

[49] Xuanqi Liu, Zhuotao Liu, Qi Li, Ke Xu, and Mingwei Xu. Pencil: Private and Extensible Collaborative Learning without the Non-Colluding Assumption. 2024.

[50] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *IEEE Symposium on Security and Privacy*, pages 1057–1073, 2021.

[51] Ming Luo, Feng-Hao Liu, and Han Wang. Faster FHE-based single-server private information retrieval. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1405–1419, 2024.

[52] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Eurocrypt*, pages 1–23, 2010.

[53] Xirong Ma, Junling Fang, Chunpeng Ge, Dung H Duong, Yali Jiang, Yanbin Li, Willy Susilo, and Lizhen Cui. New permutation decomposition techniques for efficient homomorphic permutation. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 1068–1082, 2025.

[54] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, pages 155–174, 2016.

[55] Samir Jordan Menon and David J Wu. Spiral: Fast, high-rate single-server PIR via FHE composition. In *IEEE symposium on security and privacy*, pages 930–947, 2022.

[56] Samir Jordan Menon and David J Wu. YPIR: High-Throughput Single-Server PIR with silent preprocessing. In *USENIX Security Symposium*, pages 5985–6002, 2024.

[57] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. 2011.

[58] Muhammad Haris Mughees, Hao Chen, and Ling Ren. OnionPIR: Response efficient single-server PIR. In *ACM SIGSAC conference on computer and communications security*, pages 2292–2306, 2021.

[59] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *IEEE Symposium on Security and Privacy*, pages 437–452, 2023.

[60] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. On algebraic homomorphic encryption and its applications to doubly-efficient PIR. In *EUROCRYPT*, pages 34–64. Springer, 2025.

[61] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[62] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1002–1019, 2018.

[63] Joel Reardon, Jeffrey Pound, and Ian Goldberg. Relational-complete private information retrieval. *University of Waterloo, Tech. Rep. CACR*, 34(2007), 2007.

[64] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *JACM*, 56(6):1–40, 2009.

[65] Ling Ren, Muhammad Haris Mughees, and I Sun. Simple and practical amortized sublinear private information retrieval using dummy subsets. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1420–1433, 2024.

[66] Microsoft SEAL (release 4.1.0). `https://github.com/Microsoft/SEAL`, September 2022. Microsoft Research, Redmond, WA.

[67] Radu Sion and Bogdan Carbunar. On the computational practicality of private information retrieval. In *Network and Distributed System Security Symposium*, pages 2006–06, 2007.

[68] Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71(1):57–81, 2014.

[69] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *USENIX Security Symposium*, pages 1556–1571, 2019.

[70] Edward O Thorp. Nonrandom shuffling with applications to the game of faro. *Journal of the American Statistical Association*, 68(344):842–847, 1973.

[71] Abraham Waksman. A permutation network. *Journal of the ACM (JACM)*, 15(1):159–163, 1968.

[72] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 299–313, 2017.

[73] Zhikun Wang and Ling Ren. Single-server client preprocessing PIR with tight space-time trade-off. In *EUROCRYPT*, pages 94–122. Springer, 2025.

[74] Zhou Zhang, Song Bian, Zian Zhao, Ran Mao, Haoyi Zhou, Jiafeng Hua, Yier Jin, and Zhenyu Guan. Arcedb: an arbitrary-precision encrypted database via (amortized) modular homomorphic encryption. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 4613–4627, 2024.

[75] Mingxun Zhou, Wei-Kai Lin, Yiannis Tselekounis, and Elaine Shi. Optimal single-server private information retrieval. In *EUROCRYPT*, pages 395–425. Springer, 2023.

[76] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. Piano: extremely simple, single-server PIR with sublinear server computation. In *IEEE Symposium on Security and Privacy*, pages 4296–4314. IEEE, 2024.

A Details and Proofs for RotPIR

A.1 Correctness

We present the correctness proof of Theorem 3.1 below.

Proof of correctness in Theorem 3.1. Specifically, we show that for any query index $\mathsf{idx} \in [N]$, the protocol returns the corresponding database entry $\mathbf{D}[\mathsf{idx}]$, assuming the server follows the protocol in a semi-honest manner.

Assuming the underlying homomorphic encryption scheme is correctly parameterized, the client obtains two hints at the end of the offline phase. The first hint is the vector $\mathbf{h}_j = \bigoplus_{i=0}^{\sqrt{N}-1} \mathbf{D}_i[j + \delta_i \bmod \sqrt{N}], j \in [\sqrt{N}]$ and the second hint is the vector $\mathbf{z} = (\mathbf{D}_0[r], \ldots, \mathbf{D}_{\sqrt{N}-1}[r])$ To retrieve the entry at index $\mathsf{idx} = (u, v)$, the client computes $\mathsf{ind} := v - \delta_u \bmod \sqrt{N}$ Then $\mathbf{h}_{\mathsf{ind}} = \mathbf{D}_u[v] \oplus \bigoplus_{i \in [\sqrt{N}] \setminus \{u\}} \mathbf{D}_i[\mathsf{ind} + \delta_i \bmod \sqrt{N}]$.

In the online phase, the query is constructed so that $s_i = \mathsf{ind} + \delta_i \bmod \sqrt{N}$ for $i \neq u, s_u = r$ Therefore, the server response is $\mathsf{res} = \bigoplus_{i=0}^{\sqrt{N}-1} \mathbf{D}_i[s_i] = \mathbf{D}_u[r] \oplus \bigoplus_{i \in [\sqrt{N}] \setminus \{u\}} \mathbf{D}_i[\mathsf{ind} + \delta_i \bmod \sqrt{N}]$ Since $\mathbf{z}_u = \mathbf{D}_u[r]$, the client reconstructs $\mathbf{h}_{\mathsf{ind}} \oplus \mathbf{z}_u \oplus \mathsf{res} = \mathbf{D}_u[v]$ Thus the protocol returns the desired database entry $\mathbf{D}[\mathsf{idx}]$. □

A.2 Security

We present the detailed proofs of Lemma 3.1 and Lemma 3.2 below.

Proof of Lemma 3.1. From the construction, the adversary's view in the shift game $\mathsf{SGame}^0_{\mathcal{A}}(L, K)$ differs only in the value of S^b. Therefore, it suffices to show that S^0 and S^1 are identically distributed, in which case the lemma follows.

For S^1, it is easy to see that for any fixed tuple $(p_0, p_1, \ldots, p_{L-1}) \in [K]^L$, we have that

$$\Pr\left[S^1 = (p_0, p_1, \ldots, p_{L-1})\right] = \left(\tfrac{1}{K}\right)^L.$$

We next prove that the same holds for S^0. By construction, the value s_u in S^0 is a random value r, which is uniformly distributed over $[K]$ and independent of the remaining components. Therefore, for the fixed tuple $(p_0, \ldots, p_{L-1}) \in [K]^L$, we have

$$\begin{aligned}\Pr\left[S^0 = (p_0, \ldots, p_{L-1})\right] &= \Pr[\{s_m = p_m\}_{m \neq u}] \cdot \Pr[s_u = p_u] \\ &= \tfrac{1}{K} \Pr[\{s_m = p_m\}_{m \neq u}].\end{aligned}$$

Now consider any position s_m with $m \neq u$. Since $\mathsf{ind} = v - \delta_u \pmod K$, we can write $s_m = \mathsf{ind} + \delta_m = v - \delta_u + \delta_m \pmod K$. Each δ_m is sampled independently and uniformly from $[K]$, and is independent of both v and δ_u. It follows that each s_m is uniform over $[K]$, and that all s_m for $m \neq u$ are independent. Therefore, the joint distribution of $\{s_m : m \neq u\}$ is the product of independent uniform distributions, and thus

$$\Pr[\{s_m = p_m\}_{m \neq u}] = \prod_{m \neq u} \Pr[s_m = p_m] = \left(\tfrac{1}{K}\right)^{L-1}.$$

Combining the above, we obtain

$$\Pr\left[S^0 = (p_0, \ldots, p_{L-1})\right] = \left(\tfrac{1}{K}\right)^L.$$

As a result, S^0 and S^1 are identically distributed. This completes the proof.

□

Proof of Lemma 3.2. We generalize our proofs to the case in which the database is seen as a collection of L rows each consisting of K elements, with $L \cdot K = N$. The description of our constructions set $L = K = \sqrt{N}$.

We first describe the hybrid Experiment H_0^b in Figure C1, for $b \in \{0, 1\}$. Experiment H_0^b is identical to $\mathsf{Experiment}^b_{\mathcal{A}}(\lambda, \mathbf{D})$, except that instead of encrypting the actual offsets $(\boldsymbol{\delta}, \rho)$, it encrypts independently

Experiment H_0^b

- Generate the secret key sk and the evaluation key evk for security parameter λ.
- Sample two sequences of L random values
  - $\boldsymbol{\delta} = (\delta_0, \ldots, \delta_{L-1})$ from $[K]$, $\rho \leftarrow [K]$.
  - $\boldsymbol{\delta}' = (\delta'_0, \ldots, \delta'_{L-1})$ from $[K]$, $\rho' \leftarrow [K]$.
- Encrypt the vector δ' thus obtaining $\mathsf{qu}_0 = (\mathsf{Enc}(\mathsf{sk}, \delta'_0), \ldots, \mathsf{Enc}(\mathsf{sk}, \delta'_{L-1}, \mathsf{Enc}(\mathsf{sk}, \rho'))$.
- $\mathcal{A}$ sees the evaluation key evk and ciphertexts in qu_0, outputs
  - $\mathsf{idx}_0 = (u_0, v_0)$ and $\mathsf{idx}_1 = (u_1, v_1)$
- Set $\mathsf{ind} = v_b - \delta_{u_b} \pmod K$.
- For $i \in [L] \setminus \{u_b\}$, set $s_i^b = \mathsf{ind} + \delta_i$.
- Set $s_{u_b}^b = \rho$.
- Set $S^b = (s_0^b, \ldots, s_{L-1}^b)$ and output $\mathcal{A}(S^b)$.

Appendix Figure C1: The Experiment H_0^b.

sampled offsets $(\boldsymbol{\delta}', \rho')$. Furthermore, the rest of the experiment is identical in both experiments. Since the encryption scheme is IND-CPA secure, the adversary cannot distinguish whether δ or δ' is encrypted. Therefore, for $b \in \{0, 1\}$, $\mathsf{Experiment}_{\mathcal{A}}^b(\lambda, \mathbf{D})$ and Experiment H_0^b are computationally indistinguishable.

Next, we describe hybrid Experiment H_1^b in Figure C2 that is identical to Experiment H_0^b except that instead of computing S^b from the vector $\boldsymbol{\delta}$, it samples S^b uniformly at random from $[K]^L$. First, we make the obvious observation that H_1^0 and H_1^1 are perfectly indistinguishable. Next, since the adversary only sees encryptions of $(\boldsymbol{\delta}', \rho')$, which are independent of S^b and not used elsewhere, these ciphertexts do not affect the distribution of the adversary's view. Therefore, the situation is equivalent to the shift game SGame. By Lemma 3.1, the set S^b in $\mathsf{Experiment}H_0^b$ and $\mathsf{Experiment}H_1^b$ follow the same distribution, and thus the two experiments are perfectly indistinguishable.

In conclusion, we have established the following, for each $b \in \{0, 1\}$:

1. $\mathsf{Experiment}_{\mathcal{A}}^b(\lambda, \mathbf{D})$ and hybrid Experiment H_0^b are computationally indistinguishable (by IND-CPA).
2. Hybrid Experiment H_0^b and hybrid Experiment H_1^b are perfectly indistinguishable by Lemma 3.1.
3. Hybrid Experiment H_1^0 and hybrid Experiment H_1^1 are identical.

BY (1) and (2), $\mathsf{Experiment}_{\mathcal{A}}^b(\lambda, \mathbf{D})$ and hybrid Experiment H_1^b are computationally indistinguishable. Then by (3), $\mathsf{Experiment}_{\mathcal{A}}^0(\lambda, \mathbf{D})$ and $\mathsf{Experiment}_{\mathcal{A}}^1(\lambda, \mathbf{D})$ are computationally indistinguishable.

Finally, the indistinguishability of $\mathsf{Experiment}_{\mathcal{A}}^0(\lambda, \mathbf{D})$ and $\mathsf{Experiment}_{\mathcal{A}}^1(\lambda, \mathbf{D})$ implies that the adversary's advantage in the standard security game $\mathsf{Priv}_{\mathcal{A}}^b(\lambda, \kappa, \mathbf{D}, 1)$ for Figure 4 is negligible, and hence Figure 4 satisfies the security. This completes the proof. □

B Details and Proofs for mRotPIR

In this section, we analyze the security and correctness of the protocol in Figure 7, thereby proving Theorem 4.1.

Experiment H_1^b

- Generate secret key sk and the evaluation key evk for security parameter λ.
- Sample two sequences of L random values
  - $\boldsymbol{\delta} = (\delta_0, \ldots, \delta_{L-1})$ from $[K]$, $\rho \leftarrow [K]$.
  - $\boldsymbol{\delta}' = (\delta'_0, \ldots, \delta'_{L-1})$ from $[K]$, $\rho' \leftarrow [K]$.
- Encrypt the vector δ' thus obtaining $\mathsf{qu}_0 = (\mathsf{Enc}(\mathsf{sk}, \delta'_0), \ldots, \mathsf{Enc}(\mathsf{sk}, \delta'_{L-1}))$.
- $\mathcal{A}$ sees the evaluation key evk and ciphertexts in qu_0, outputs
  - $\mathsf{idx}_0 = (u_0, v_0)$ and $\mathsf{idx}_1 = (u_1, v_1)$
- For $i \in [L] \setminus \{u_b\}$, randomly sample $s_i^b \leftarrow [K]$.
- Randomly select $s_{u_b}^b$ from $[K]$.
- Set $S^b = (s_0^b, \ldots, s_{L-1}^b)$ and output $\mathcal{A}(S^b)$.

Appendix Figure C2: The Experiment H_1^b.

B.1 Correctness

The correctness of mRotPIR builds on the correctness of the underlying single-query protocol RotPIR, as established in Theorem 3.1. Consequently, the only case in which mRotPIR fails is when the client cannot find an unused hint for the chunk associated with a query. For such a failure to occur over a sequence of Q queries, at least $\ell + 1$ of these queries must be assigned to the same chunk. The following lemma indicates that this probability is negligible in κ. Therefore, protocol mRotPIR returns the correct results with probability at least $1 - \mathsf{negl}(\kappa)$.

Lemma B.1 *Consider the partition of a size N database in Q chunks of size N/Q computed using a PRP with security parameter λ. For any sequence $(\mathsf{idx}_1, \ldots, \mathsf{idx}_Q)$ of Q queries, the probability that at least $\ell + 1$ land to the same chunk is negligible in κ.*

To prove Lemma B.1 We prove the following lemma that the entries of the database are partitioned in Q chunks of size $C = N/Q$.

Lemma B.2 *Let $\mathcal{Q}$ be any sequence of Q indices from $[N]$ and suppose that the N entries of the database are randomly partitioned into Q chunks each containing exactly $C = N/Q$ entries. Then the probability that more than ℓ queries from $\mathcal{Q}$ fall into the same chunk is negligible in κ.*

Proof. We remind the reader that the client keeps a cache of the results of the past queries and thus no repeated query is performed. Indeed, when the index being queried is found in the cache, the client performs a query for a *target* index that is not in the cache. Therefore without loss of generality we fix a sequence $\mathcal{Q} = (\mathsf{idx}_0, \ldots, \mathsf{idx}_{Q-1})$ of Q *distinct* queries and we let I be a subset $I \subseteq \mathcal{Q}$ of size $\ell + 1$. We bound the probability of event $\mathtt{Bad}(I, c)$ that all $\ell+1$ queries of I fall into a given chunk c and $\mathtt{Bad}(c) = \bigcup_{I \subseteq \mathcal{Q}} \mathtt{Bad}(I, c)$. The probability is taken over the assignment of indices to chunks. Note that in the correctness game, the sequence $\mathcal{Q}$ is fixed before the partition of the indices into chunks which is performed randomly and independently from $\mathcal{Q}$. We also assume that $\ell < N/Q$, since the number of hints in each chunk is at most the number of distinct entries in a chunk.

Since the database is randomly permuted and evenly partitioned into Q chunks of size N/Q, the probability that, for $i = 0, \ldots, \ell$, the i-th query of I falls into chunk c given that the previous $i - 1$ have been

assigned to chunk c, is

$$\frac{N/Q - i}{N - i} \leq \frac{N/Q}{N - N/Q} = \frac{1}{Q-1} \qquad (\text{since } i \leq \ell < N/Q).$$

Therefore, by the multiplication rule, the probability that *all* $\ell + 1$ queries of I fall into chunk c is

$$\Pr[\mathtt{Bad}(I, c)] \;=\; \prod_{i=0}^{\ell} \frac{N/Q - i}{N - i} \;\leq\; \left(\frac{1}{Q-1}\right)^{\ell+1}.$$

Taking a union bound over all $\binom{Q}{\ell+1}$ subsets I, the probability that *some* subset of $\ell + 1$ queries falls into the chunk c is at most

$$\Pr[\mathtt{Bad}(c)] \;\leq\; \binom{Q}{\ell+1} \cdot \left(\frac{1}{Q-1}\right)^{\ell+1} \tag{A1}$$

We now bound Equation (A1) more carefully. Using $\binom{Q}{\ell+1} = \dfrac{Q(Q-1)\cdots(Q-\ell)}{(\ell+1)!} \leq \dfrac{Q^{\ell+1}}{(\ell+1)!}$, we obtain

$$\Pr[\mathtt{Bad}(c)] \;\leq\; \frac{1}{(\ell+1)!} \cdot \left(\frac{Q}{Q-1}\right)^{\ell+1}.$$

Applying $1 + x \leq e^x$ with $x = \frac{1}{Q-1}$,

$$\left(\frac{Q}{Q-1}\right)^{\ell+1} = \left(1 + \frac{1}{Q-1}\right)^{\ell+1} \leq \exp\!\left(\frac{\ell+1}{Q-1}\right),$$

so that

$$\Pr[\mathtt{Bad}(c)] \;\leq\; \frac{1}{(\ell+1)!} \cdot \exp\!\left(\frac{\ell+1}{Q-1}\right).$$

which is negligible in ℓ since $(\ell+1)!$ grows super-exponentially and $\exp\!\left(\frac{\ell+1}{Q-1}\right) \leq e^3$ for $\ell \leq Q$. Taking a union bound over all Q chunks completes the proof. □

Comparison with Prior Bound. The prior analysis of [20] establishes the same result via the bound $\left(\frac{6}{\ell}\right)^{\ell}$, which is also negligible in ℓ. Table A1 compares the two bounds for the standard parameter setting $Q = N^{1/3} = 2^{10}$ (for $N = 2^{30}$):

Appendix Table A1: Correctness failure probability bounds for $Q = 2^{10}$.

ℓ	Prior Bound [20]	Proof of Lemma B.2
16	$\approx 2^{-12.64}$	$\approx 2^{-38.3}$
17	$\approx 2^{-15.54}$	$\approx 2^{-42.5}$

Our bound is tighter. To achieve a standard correctness target of 2^{-40}, the prior bound requires $\ell = 25$, whereas our analysis shows that $\ell = 17$ already suffices. Since hint storage, offline communication, and server preprocessing all scale linearly in ℓ, this translates to a direct reduction in the overhead of mRotPIR at no loss to the correctness guarantee. In our implementation, we set $\ell = 17$.

Hybrid Privacy Game $\mathsf{HPriv}^t_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q)$.
Parametrized by an adversary $\mathcal{A}$ and integer $t \in \{0, \ldots, Q\}$.

- $(\mathsf{st}, \mathsf{qu}_0) \leftarrow \Pi.\mathsf{HintQuery}(1^\lambda, 1^\kappa, 1^{|\mathbf{D}|})$.
- $\mathsf{res}_0 \leftarrow \Pi.\mathsf{HintAnswer}(\mathbf{D}, \mathsf{qu}_0)$.
- $\mathsf{hint} \leftarrow \mathsf{HintExtract}(\mathsf{st}, \mathsf{res}_0)$.
- $\mathsf{Advst} \leftarrow \mathcal{A}(1^\lambda, \mathsf{qu}_0, \mathsf{res}_0)$.
- For $l = 1, ..., t$:
  - $(\mathsf{Advst}, \mathsf{idx}^0_l, \mathsf{idx}^1_l) \leftarrow \mathcal{A}(\mathsf{Advst})$
  - $(\mathsf{st}', \mathsf{qu}_l) \leftarrow \Pi.\mathsf{Query}(\mathsf{st}, \mathsf{idx}^0_l)$
  - $\mathsf{Advst} \leftarrow \mathcal{A}(\mathsf{Advst}, \mathbf{D}, \mathsf{qu})$
- For $l = t+1, ..., Q$:
  - $(\mathsf{Advst}, \mathsf{idx}^0_l, \mathsf{idx}^1_l) \leftarrow \mathcal{A}(\mathsf{Advst})$
  - $(\mathsf{st}', \mathsf{qu}_l) \leftarrow \Pi.\mathsf{Query}(\mathsf{st}, \mathsf{idx}^1_l)$
  - $\mathsf{Advst} \leftarrow \mathcal{A}(\mathsf{Advst}, \mathbf{D}, \mathsf{qu})$
- Output $b \leftarrow \mathcal{A}(\mathsf{Advst})$

Appendix Figure C3: The Hybrid Privacy Game $\mathsf{HPriv}^t_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q)$.

B.2 Security

To prove security, we design a sequence of $Q+1$ hybrid games $\mathsf{HPriv}^t_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q)$, for $t \in [Q]$ in Figure C3. For every adversary $\mathcal{A}$, $\mathsf{HPriv}^0_{\mathcal{A}}$ is the security game $\mathsf{Priv}^1_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q)$ defined in Figure 3, and $\mathsf{HPriv}^Q_{\mathcal{A}}$ is the security game $\mathsf{Priv}^0_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q)$ defined in Figure 3. The security requirement is therefore equivalent to proving that, for any PPT adversary $\mathcal{A}$,

$$\begin{aligned} &| \Pr[\mathsf{HPriv}^0_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q) \to 1] - \\ &\quad \Pr[\mathsf{HPriv}^Q_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q) \to 1] | \leq \mathsf{negl}(\lambda) \end{aligned} \tag{A2}$$

This claim follows directly from the following lemma.

Lemma B.3 *For all PPT adversaries $\mathcal{A}$, and for all databases $\mathbf{D}$ of size N polynomial in λ, for $t = 0, 1, \ldots, Q-1$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:*

$$\begin{aligned} &| \Pr[\mathsf{HPriv}^t_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q) \to 1] - \\ &\quad \Pr[\mathsf{HPriv}^{t+1}_{\mathcal{A}}(\lambda, \kappa, \mathbf{D}, Q) \to 1] | \leq \mathsf{negl}(\lambda). \end{aligned} \tag{A3}$$

Proof of Lemma B.3. In hybrid game HPriv^t, the first t queries are made using indices $\mathsf{idx}^0_1, \ldots, \mathsf{idx}^0_t$, while the remaining queries use $\mathsf{idx}^1_{t+1}, \ldots, \mathsf{idx}^1_Q$. In contrast, in game HPriv^{t+1}, the first $t+1$ queries use $\mathsf{idx}^0_1, \ldots, \mathsf{idx}^0_{t+1}$, and the rest use $\mathsf{idx}^1_{t+2}, \ldots, \mathsf{idx}^1_Q$. Thus, the only difference between the two games lies in the $(t+1)$-st query. Specifically, in HPriv^t the $(t+1)$-st query of the underlying single query PIR Π is for the target index tidx^1, whereas in HPriv^{t+1} it is for the target index tidx^0. By the way the target indices are constructed, each query is associated with a fresh hint and therefore we can leverage the security of Π.

Suppose, for sake of contradiction, that there exists a PPT adversary $\mathcal{A}$ that distinguishes HPriv^t from HPriv^{t+1}. We construct a PPT adversary $\mathcal{B}$ that violates the security of Π as follows.

$\mathcal{B}$ runs $\mathcal{A}$ and simulates for it the privacy game for mRotPIR, while it is playing the privacy game for Π. Specifically, for each pair $(\mathsf{idx}^0_i, \mathsf{idx}^1_i)$ output by $\mathcal{A}$, for $i \neq t+1$, $\mathcal{B}$ selects idx^0_i or idx^1_i, according to whether

$i \leq t$ or $i > t+1$, constructs the target index tidx_i and constructs the query qu_i for the target index to be submitted to $\mathcal{A}$ (see Figure 3).

Query $t+1$ is instead treated differently. $\mathcal{B}$ receives the pair $(\mathsf{idx}^0_{t+1}, \mathsf{idx}^1_{t+1})$ from $\mathcal{A}$, constructs the target indices $(\mathsf{tidx}^0_{t+1}, \mathsf{tidx}^1_{t+1})$ and submits the pair to the challenger of the privacy game for Π, receiving qu_{t+1} as a reply. $\mathcal{B}$ forwards qu_{t+1} to $\mathcal{A}$.

Finally, $\mathcal{B}$ outputs the same output given by $\mathcal{A}$.

Now it is easy to see that if $\mathcal{B}$ is playing the privacy game for mRotPIR with $b = 0$, then it has simulated the hybrid game HPriv^{t+1} for $\mathcal{A}$. On the other hand, if $\mathcal{B}$ is playing the privacy game for mRotPIR with $b = 1$, then it has simulated the hybrid game HPriv^{t} for $\mathcal{A}$. Since $\mathcal{A}$ is assumed to distinguish the two hybrid games, it follows that $\mathcal{B}$ breaks the security of Π. Contradiction.

□

Based on Lemma B.3, the experiments $\mathsf{HPriv}^t_{\mathcal{A}}$ and $\mathsf{HPriv}^{t+1}_{\mathcal{A}}$ are computationally indistinguishable and by the triangle inequality, we obtain

$$\left|\Pr[\mathsf{HPriv}^0_{\mathcal{A}}(\lambda, \mathbf{D}, Q) = 1] - \Pr[\mathsf{HPriv}^Q_{\mathcal{A}}(\lambda, \mathbf{D}, Q) = 1]\right| \leq Q \cdot \mathsf{negl}(\lambda).$$

which is negligible in λ since Q is polynomially bounded. This proves Equation (A2) and completes the security analysis.

C Full Notations, Ciphertext Type, and Operators

We summarize the notations, ciphertext type, and homomorphic operators used in this work in Table A2 and Table A3.

• $\overline{\mathsf{RLWE}}^{n,q}_{\tilde{s}}(\tilde{m})$: Given a gadget vector $\mathbf{g} = (1, \ldots, B^{L-1})$, the extend RLWE encryption of a message $\tilde{m} \in R_P$ under the secret key $\tilde{s} \in R_q$ is defined as:

$$\overline{\mathsf{RLWE}}^{n,q}_{\tilde{s}}(\tilde{m}) = \{\mathsf{RLWE}(g_i \cdot \tilde{m})\}_{i=0}^{L-1} \tag{A4}$$

The concrete constructions of the $\overline{\mathsf{RLWE}}$ ciphertext can be found in [25, 43]. We can view an $\overline{\mathsf{RLWE}}$ ciphertext as a tuple of L standard RLWE ciphertexts. We use $\overline{\mathsf{RLWE}}_i / \overline{\mathsf{RLWE}}[i]$ to denote the i-th RLWE ciphertext of $\overline{\mathsf{RLWE}}$.

• $\mathsf{HEAuto}(\mathsf{RLWE}(\tilde{m}), t, \mathsf{evk})$: Given an RLWE ciphertext $\mathsf{RLWE}(\tilde{m})$ with coefficient encoding, a mapping index t, and an evaluation key evk, the homomorphic automorphism HEAuto outputs a ciphertext $\mathsf{RLWE}(\tau_t(\tilde{m}))$ where $\tau_t(X) = X^t$. More details on HEAuto can be found in [29, 17]. Similarly, we define the homomorphic automorphism on $\overline{\mathsf{RLWE}}$ ciphertext and $\mathsf{HEAuto}(\overline{\mathsf{RLWE}}(\tilde{m}), t, \mathsf{evk})$ outputs $\overline{\mathsf{RLWE}}(\tau_t(\tilde{m}))$.

• $\diamond$: $\mathsf{InnerProduct}$ is a type of homomorphic multiplication between polynomial $\tilde{r}$ and $\overline{\mathsf{RLWE}}$ ciphertext. The multiplication first decompose $\tilde{r} = \sum_{l=0}^{L-1} g_l \cdot \tilde{r}_l$ and then evaluate $\tilde{r} \diamond \overline{\mathsf{RLWE}}(\tilde{m}) = \sum_{l=0}^{L-1} \tilde{r} \cdot \overline{\mathsf{RLWE}}(\tilde{m})[l] = \mathsf{RLWE}(\tilde{r} \cdot \tilde{m})$.

C.1 Noise Analysis of the Operator

Suppose the extend RLWE ciphertext is instantiated with a dimension n, decomposition base B and length L. the noise variance of the $\diamond$ is $\sigma^2_{\odot} \leq \frac{1}{12} nlB^2 \sigma^2_{\mathsf{in}}$, where σ^2_{in} is the noise variance of the input extend RLWE ciphertext.

Similarly, the output noise of the operator $\mathsf{HERotate}$ $\sigma^2_{\mathsf{HERotate}} \leq \frac{1}{12} nlB^2 \sigma^2_{\mathsf{evk}} + \sigma^2_{in}$ where σ^2_{evk} is the noise variance of the evaluation key and σ^2_{in} is the noise variance of the input RLWE ciphertext. Moreover, the noise variance of $\boxdot$ is $\sigma^2_{\boxdot} \leq 2\sigma^2_{\odot} + b\sigma^2_{in} = \frac{1}{6} nlB^2 \sigma^2_{\mathsf{RGSW}} + ||\tilde{m}||^2_2 \sigma^2_{\mathsf{RLWE}}$ where $\tilde{m}$ is the plaintext of the RGSW ciphertext.

Appendix Table A2: Summary of Notations

Notation	Description
$\mathbf{D}$	The database
N	The database size
st	Internal client state
qu_0	Hint query
res_0	Hint response
idx	Database index
qu	Online query
res	Online response
val	Retrieval result
λ	The security parameter
p	The plaintext modulus
q	The ciphertext modulus
$\mathbb{Z}_q$	The ring of integers modulo q
$\mathbb{Z}[X]$	The polynomial ring over $\mathbb{Z}$
R	The cyclotomic ring $\mathbb{Z}[X]/(X^N+1)$
R_q	The cyclotomic ring $\mathbb{Z}_q[X]/(X^N+1)$
χ	The noise distribution
Δ	The scaling factor
$\mathbf{a}/\tilde{a}/\mathbf{A}$	A vector/polynomial/matrix
$\mathbf{a}_i$	The i-th element of $\mathbf{a}$
$\tilde{a}_i$	The i-th coefficient of $\tilde{a}$
$\mathbf{A}_i$	The i-th row of $\mathbf{A}$
$\mathsf{RLWE}_{\tilde{s}}^{n,q}(\tilde{m})$	An RLWE ciphertexts encrypting $\tilde{m}$ with parameters (n,q) and secret $\tilde{s}$
$\overline{\mathsf{RLWE}}_{\tilde{s}}^{n,q}(\tilde{m})$	An extend RLWE ciphertexts encrypting $\tilde{m}$ with parameters (n,q) and secret $\tilde{s}$
$\mathsf{RGSW}_{\tilde{s}}^{n,q}(\tilde{m})$	An RGSW ciphertext encrypting $\tilde{m}$ with parameters (n,q) and secret $\tilde{s}$

D Details of Oblivious Homomorphic Rotation

D.1 Additional Lemma and Proofs

Lemma D.1 (Correctness for Algorithm 1.) *Suppose the RGSW ciphertexts and the evaluation key are instantiated with ring dimension n, decomposition base B, and decomposition length L. For the input vector $\mathbf{m}$, Algorithm 1 outputs an RLWE ciphertext ct_{out} encrypting $\mathbf{m} \ll \delta$ with noise variance $Var(ct_{\mathsf{out}}) \leq \frac{1}{12} n \log n l B^2 (2\sigma^2_{\mathsf{RGSW}} + \sigma^2_{\mathsf{evk}})$ where σ^2_{RGSW} and σ^2_{evk} denote the noise variances of a fresh RGSW ciphertext and the evaluation key, respectively.*

Proof of Lemma D.1. Suppose the extend RLWE and RGSW ciphertexts and the evaluation key are instantiated with a dimension n, decomposition base B and length L. We first introduce the noise variance of the basic HE operator defined in Section 2.2.2. The output noise of the polynomial with an extend RLWE ciphertext is is $\sigma^2_{\odot} \leq \frac{1}{12} n l B^2 \sigma^2_{\mathsf{in}}$, where σ^2_{in} is the noise variance of the input extend RLWE cipher-

Appendix Table A3: Summary of Operators

Notation	Description
HintQuery	Client-side hint query generation
HintAnswer	Server-side hint response computation
HintExtract	Client-side hint extraction and state update
Query	Client-side online query generation
Answer	Server-side online response computation
ReConstruct	Client-side reconstruction of $\mathbf{D}[\mathsf{idx}]$
HEAuto	Homomorphic automorphism [29]
HERotate	Homomorphic rotation [28]
$\odot$	Component-wise (slot-wise) multiplication
$\diamond$	Inner Product
$\boxdot$	ExternalProduct
ObHERotate	Oblivious homomorphic rotation Algorithm 1
QueryPack	Query pack Algorithm 4
CtExpand	Oblivious homomorphic rotation Algorithm 5
QueryUnpack	Oblivious homomorphic rotation Algorithm 6

text. Similarly, the output noise of the operator HERotate $\sigma^2_{\mathsf{HERotate}} \leq \frac{1}{12} nlB^2\sigma^2_{\mathsf{evk}} + \sigma^2_{in}$ where σ^2_{evk} is the noise variance of the evaluation key and σ^2_{in} is the noise variance of the input RLWE ciphertext. Moreover, the noise variance of $\boxdot$ is $\sigma^2_{\boxdot} \leq 2\sigma^2_{\odot} + b\sigma^2_{in} = \frac{1}{6} nlB^2\sigma^2_{\mathsf{RGSW}} + ||\tilde{m}||^2_2\sigma^2_{\mathsf{RLWE}}$ where $\tilde{m}$ is the plaintext of the RGSW ciphertext. In each iteration of the Algorithm 1, $ct_{i+1} = (ct_r - ct_i) \boxdot \mathsf{RGSW}(\delta_b) + ct_i$, since $\delta_b = 0/1$, the noise variance $Var(ct_{i+1}) \leq \frac{1}{6} nlB^2\sigma^2_{\mathsf{RGSW}} + \frac{1}{12} nlB^2\sigma^2_{\mathsf{evk}} + \sigma^2_{in}$ Therefore, the output ciphertext $Var(ct_{\mathsf{out}}) \leq \frac{1}{12} n \log nlB^2(2\sigma^2_{\mathsf{RGSW}} + \sigma^2_{\mathsf{evk}}) + Var(ct_0) = \frac{1}{12} n \log nlB^2(2\sigma^2_{\mathsf{RGSW}} + \sigma^2_{\mathsf{evk}})$

□

Lemma D.2 (Correctness for Algorithm 2.) *Suppose the RGSW ciphertexts and the evaluation key are instantiated with ring dimension n, decomposition base B, and decomposition length L. For the input vector $\mathbf{m}$ with length $N = K \cdot n$, Algorithm 2 outputs K RLWE ciphertexts $ct = \{ct_0, ct_1, \cdots, ct_{K-1}\}$ where ct_i encrypts $(\mathbf{m} << \delta)_{[i\cdot n:(i+1)\cdot n]}$ with noise variance $\sigma^2_{\mathsf{MulObHERotate}} = 2\sigma^2_{\odot}(\sigma^2_{\mathrm{in}1}, \sigma^2_{\mathrm{in}2})$ where $\sigma^2_{\mathrm{in}1} \leq \frac{1}{6} nlB^2 \lceil \log K \rceil \sigma^2_{\mathsf{RGSW}} + \sigma^2_{\mathsf{ObHERotate}}$, $\sigma^2_{\mathrm{in}2} = \sigma^2_{\mathsf{ObHERotate}}$, and $\sigma^2_{\odot}$ is the noise variance of $\odot$.*

Proof of Lemma D.2. In the cross-ciphertext rotation, the dominant noise growth comes from the $\boxdot$ operation, its noise variance can be bounded as

$$\sigma^2_{\boxdot} \;\leq\; 2\sigma^2_{\odot} \;+\; \delta_t^{\mathsf{cross}}\,\sigma^2_{\mathrm{in}} \;=\; \frac{1}{6}\,nlB^2\,\sigma^2_{\mathsf{RGSW}} \;+\; \|\delta_t^{\mathsf{cross}}\|_2^2\,\sigma^2_{\mathsf{RLWE}},$$

where σ^2_{in} denotes the input ciphertext noise variance and $\delta_t^{\mathsf{cross}} \in \{0, 1\}$ is the t-th (encrypted) selector bit. Since $\delta_t^{\mathsf{cross}}$ is binary, $\|\delta_t^{\mathsf{cross}}\|_2^2 = \delta_t^{\mathsf{cross}}$. Thus, each iteration of Line 12 contributes an additional $\frac{1}{6} nlB^2\sigma^2_{\mathsf{RGSW}}$ to the variance bound. After processing all $\lceil \log K \rceil$ bits, the noise variance of the ciphertext sequence $ct = \{ct_0, ct_1, \ldots, ct_{K-1}\}$ is bounded by

$$Var(ct) \;\leq\; \frac{1}{6}\,nlB^2\,\lceil \log K \rceil\,\sigma^2_{\mathsf{RGSW}}.$$

The inner-ciphertext rotation invokes ObHERotate, hence after Lines 13–15 the noise variance satisfies

$$Var(ct) \;\leq\; \frac{1}{6}\,nlB^2\,\lceil \log K \rceil\,\sigma^2_{\mathsf{RGSW}} \;+\; \sigma^2_{\mathsf{ObHERotate}}.$$

Algorithm 4: QueryPack

Input : A set of rotation index $\{\delta_0, \delta_1, ..., \delta_{n-1}\}$ where $\delta_k \in [n]$.
Output: $\log n$ extend RLWE ciphertexts packing the indices: $\mathsf{C}_{\overline{\mathsf{RLWE}}} = \{\overline{\mathsf{RLWE}}(pt_b)\}$.

Depompose δ_k as $\delta_k = \sum_{b=0}^{\log n-1} \delta_{k,b} \cdot 2^b$
for $b = 0$ **to** $\log n - 1$ **do**
  $pt_b \leftarrow \delta_{0,b} + \delta_{1,b}X + \cdots + \delta_{n-1,b}X^{n-1}$
  $\mathsf{C}_{\overline{\mathsf{RLWE}}}[i] \leftarrow \overline{\mathsf{RLWE}}(n^{-1} \cdot pt_b)$
Return: $\mathsf{C}_{\overline{\mathsf{RLWE}}}$

Algorithm 5: CtExpand

Input : An extend RLWE cipher $ct = \overline{\mathsf{RLWE}}(n^{-1}\tilde{m})$.
Input : An evaluation key evk for automorphism.
Output: n extend RLWE ciphertext $ct_i = \overline{\mathsf{RLWE}}(\tilde{m}_i)$.

$ct_0 \leftarrow ct$
for $i = 0$ **to** $\log n - 1$ **do**
  for $j = 0$ **to** $2^i - 1$ **do**
    $ct_{auto} \leftarrow \mathsf{HEAuto}(ct_j, n/2^i + 1, \mathsf{evk}_i)$
    $ct_{j+2^i} \leftarrow (ct_j - ct_{auto}) \cdot X^{-2^i}$
    $ct_j \leftarrow ct_j + ct_{auto}$
Return: $ct_0, ct_1, ..., ct_{n-1}$

Similarly, since ct_{tail} (and thus $ct_{\mathsf{head}} = \mathbf{1} - ct_{\mathsf{tail}}$) is constructed via the same ObHERotate-style procedure, we have

$$Var(ct_{\mathsf{tail}}) = Var(ct_{\mathsf{head}}) \;=\; \sigma^2_{\mathsf{ObHERotate}}.$$

Finally, the cross-block propagation computes $ct_b \leftarrow ct_b \odot ct_{\mathsf{head}} + ct_{(b+1) \bmod B} \odot ct_{\mathsf{tail}}$, i.e., the sum of two slot-wise multiplications. Let

$$\sigma^2_{\mathrm{in}1} \le \frac{1}{6}\, nlB^2 \lceil \log K \rceil\, \sigma^2_{\mathsf{RGSW}} + \sigma^2_{\mathsf{ObHERotate}}, \sigma^2_{\mathrm{in}2} = \sigma^2_{\mathsf{ObHERotate}}.$$

Then the resulting variance is bounded by

$$\sigma^2_{\mathsf{MulObHERotate}} = 2\sigma^2_{\odot}(\sigma^2_{\mathrm{in}1}, \sigma^2_{\mathrm{in}2})$$

. □

D.2 Query Compression

Hint Query Packing. Instead of encrypting each query bit individually as a ciphertext, we pack the query bits corresponding to distinct data vectors into a single plaintext polynomial. The detailed procedure of QueryPack is formalized in Algorithm 4. Concretely, let $\delta_0, \delta_1, \ldots, \delta_{n-1}$ denote the rotation indices for the n data rows. For each bit position $b \in [\log n]$, the client forms a plaintext polynomial $p_b(X) = \delta_{0,b} + \delta_{1,b}X + \cdots + \delta_{n-1,b}X^{n-1}$ where $\delta_{k,b}$ is the b-th bit of δ_k. After that, each polynomial $p_b(X)$ is encrypted into an extended RLWE ciphertext and results in $\log n$ extended RLWE ciphertexts, serving as the hint query sent to the server.

Hint Query Unpacking. Once receiving the packed hint query consisting of $\log n$ extended RLWE ciphertexts, the server unpacks the query into $n \log n$ RGSW ciphertexts. To achieve this, we first introduce the ciphertext expansion operator CtExpand, which serves as the key building block for the query unpack. As shown in Algorithm 5, the ciphertext expansion operator CtExpand takes as input an extended RLWE ciphertext $ct = \overline{\mathsf{RLWE}}(n^{-1} \cdot \tilde{m})$ and outputs n extend RLWE ciphertexts $ct_i = \overline{\mathsf{RLWE}}(\tilde{m}_i)$, each encrypting one coefficient of $\tilde{m}$. The procedure basically follows the automorphism-based expansion paradigm proposed in [16], and we refer readers to [16] for detailed explanation of the CtExpand.

Algorithm 6: QueryUnpack

Input : $\log n$ extend RLWE ciphertexts packing the hint query $\mathsf{C}_{\overline{\mathsf{RLWE}}} = \{\overline{\mathsf{RLWE}}(pt_b)\}$.
Input : An evaluation key evk for automorphism.
Input : An RGSW ciphertext $\mathsf{RGSW}(\tilde{s})$.
Output: RGSW vectors encrypting n indices $\{\delta_k\}_{k=0}^{n-1}$: $\{\mathsf{C}^k_{\mathsf{RGSW}}\}_{k=0}^{n-1}, \mathsf{C}^k_{\mathsf{RGSW}} = \{\mathsf{RGSW}(\delta_{k,b})\}_{b=0}^{\log n-1}$ and $\delta_k = \sum_{b=0}^{\log n-1} \delta_{k,b} \cdot 2^b$

1 **for** $b = 0$ **to** $\log n - 1$ **do**
2 $\quad \{ct_k\}_{k=0}^{n-1} \leftarrow \mathsf{CtExpand}(\overline{\mathsf{RLWE}}(pt_b), \mathsf{evk})$
3 $\quad$ **for** $k = 0$ **to** $n-1$ **do**
4 $\quad\quad \mathsf{C}^k_{\mathsf{RGSW}}[b][0] \leftarrow ct_k$
5 $\quad\quad \mathsf{C}^k_{\mathsf{RGSW}}[b][1] \leftarrow ct_k \boxdot \mathsf{RGSW}(\tilde{s})$

Return: $\{\mathsf{C}^k_{\mathsf{RGSW}}\}_{k=0}^{n-1}$

Leveraging $\mathsf{CtExpand}$ as a subroutine, the $\mathsf{QueryUnpack}$ algorithm reconstructs the RGSW ciphertexts from the packed extend RLWE ciphertexts. As indicated in Algorithm 6, the input consists of $\log n$ extended RLWE ciphertexts $\{\overline{\mathsf{RLWE}}(pt_b)\}_{b=0}^{\log n-1}$, each encrypting the b-th bits of the rotation indices $\{\delta_{k,b}\}_{k=0}^{n-1}$. For each bit position $b \in [\log n]$, first on Line 2, the algorithm applies $\mathsf{CtExpand}$ to $\overline{\mathsf{RLWE}}(pt_b)$, and produces n extended RLWE ciphertexts $\{ct_k\}_{k=0}^{n-1}$ encrypting the individual bits $\delta_{k,b}$. Next on Line 3-4, each ct_k is placed into the first component of an RGSW ciphertext. To obtain the second component, the algorithm computes the external product $ct_k \boxdot \mathsf{RGSW}(\tilde{s})$ on Line 5, resulting in an extend RLWE ciphertext that encrypts $\delta_{k,b} \cdot \tilde{s}$. Together, these two components form $\mathsf{RGSW}(\delta_{k,b}) = (\overline{\mathsf{RLWE}}(\delta_{k,b}), \overline{\mathsf{RLWE}}(\delta_{k,b}\tilde{s}))$. Iterating over all $b \in [\log n]$ and $k \in [n]$, the algorithm reconstructs n RGSW vectors $\{\mathsf{C}^k_{\mathsf{RGSW}}\}_{k=0}^{n-1}$. Each vector $\mathsf{C}^k_{\mathsf{RGSW}}$ consists of the bit-wise RGSW encryptions of the index δ_k, namely $\mathsf{C}^k_{\mathsf{RGSW}} = \{\mathsf{RGSW}(\delta_{k,b})\}_{b=0}^{\log n-1}$ and $\delta_k = \sum_{b=0}^{\log n-1} \delta_{k,b} \cdot 2^b$. We formalize the correctness analysis of $\mathsf{QueryUnpack}$ in Lemma D.3 and the detailed proof is given in Section D.2.

Lemma D.3 (Correctness for Algorithm 6.) *Suppose the RGSW ciphertexts and the evaluation key are instantiated with a decomposition base B and length L. Algorithm 6 outputs the RGSW ciphertexts which encrypts the bits of the rotation index and for each RGSW ciphertext $\mathsf{RGSW}(\delta_{k,b})$, it follows $Var(ct_{\mathsf{out}}) \leq \frac{1}{6}nlB^2\sigma^2_{\mathsf{RGSW}} + ||\tilde{s}||_2^2\sigma^2_{\mathsf{CtExpand}}$ where σ^2_{RGSW} is the noise variance of $\mathsf{RGSW}(\tilde{s})$ and $\sigma^2_{\mathsf{CtExpand}} = n^2Var(ct_{\mathsf{in}}) + \frac{n^2-1}{3} \cdot \frac{1}{12}nlB^2\sigma^2_{\mathsf{evk}}$. Here, σ^2_{evk} is the noise variance of the evaluation key and $Var(ct_{\mathsf{in}})$ is the noise variance of the input extend RLWE ciphertext.*

Lemma D.4 (Correctness for Algorithm 3.) *Suppose the RGSW ciphertexts and the evaluation key are instantiated with ring dimension n, decomposition base B, and decomposition length L. For K input vectors $\{\mathbf{m}_0, \mathbf{m}_1, ..., \mathbf{m}_{K-1}\}$ of length N and $n = K \cdot N$, Algorithm 2 outputs one RLWE ciphertext ct encrypting $\mathbf{m}_0 \ll \delta_0 \| \mathbf{m}_1 \ll \delta_1 \| \cdots \| \mathbf{m}_{K-1} \ll \delta_{K-1}$ with noise variance $\sigma^2_{\mathsf{PackObHERotate}} = Knp^2\sigma^2_{\mathsf{ObHERotate}}$ where p is the plaintext modulus of the RLWE ciphertext and $\sigma^2_{\mathsf{ObHERotate}}$ is the noise variance of $\mathsf{ObHERotate}$.*

Proof of Lemma D.3. Suppose the RGSW ciphertexts and the evaluation key are instantiated with a decomposition base B and length L. As shown in [16], Algorithm 5 outputs the RLWE ciphertexts with noise variance $Var(ct_{\mathsf{out}}) \leq n^2Var(ct_{\mathsf{in}}) + \frac{n^2-1}{3}\sigma^2_{\odot} = n^2Var(ct_{\mathsf{in}}) + \frac{n^2-1}{3} \cdot \frac{1}{12}nlB^2\sigma^2_{\mathsf{evk}}$. Based on $\mathsf{CtExpand}$, Algorithm 6 outputs the RGSW ciphertexts which encrypts the bits of the rotation index and for each RGSW ciphertext $\mathsf{RGSW}(\delta_{k,b})$, it follows $Var(\mathsf{RGSW}_0) \leq \sigma^2_{\mathsf{CtExpand}}$, and the $Var(\mathsf{RGSW}_1) \leq 2\sigma^2_{\odot} + ||\tilde{s}||_2^2\sigma^2_{\mathsf{CtExpand}}$. In short, the output ciphertext satisfies $Var(ct_{\mathsf{out}}) \leq \frac{1}{6}nlB^2\sigma^2_{\mathsf{RGSW}} + ||\tilde{s}||_2^2\sigma^2_{\mathsf{CtExpand}}$ □