PIRCOR: Communication-Optimal Hintless Single-Server PIR via Homomorphic Rotation

Xue Yang*
xueyang@swjtu.edu.cn
Southwest Jiaotong University
Chengdu, Sichuan, China

Ruida Wang*
wangruida@iie.ac.cn
School of Cyber Security, UCAS
Beijing, China

Depan Peng
pengdepan@my.swjtu.edu.cn
Southwest Jiaotong University
Chengdu, Sichuan, China

Kun Liu
liukun@iie.ac.cn
School of Cyber Security, UCAS
Beijing, China

Xianhui Lu†
luxianhui@iie.ac.cn
School of Cyber Security, UCAS
Beijing, China

Xiaohu Tang†
xhutang@swjtu.edu.cn
Southwest Jiaotong University
Chengdu, Sichuan, China

Abstract

This work addresses the hintless single-server Private Information Retrieval (PIR) from the perspective of high-level protocol design and introduces PIRCOR and PIRCOR* that outperform the state-of-the-art PIRANA (Liu et. al., *IEEE S&P* 2024) and YPIR (Menon and Wu, *USENIX Security* 2024) in terms of the query size and the query generation time. In PIRCOR, we construct an efficient *Rotation-based Expanded Binary Code* (REBC) to expand α primary codewords into β expanded codewords by the *Rotation-Mutual-Multiplication* operation. By leveraging the innovative REBC, PIRCOR reduces the query size for single-query PIR by a factor of $O\left(N^{\frac{\delta-1}{\delta}}\right)$ compared to PIRANA, while also avoiding the $O(N + \frac{|\mathrm{DB}|}{N})$ linear scaling inherent in YPIR (N, δ and $|\mathrm{DB}|$ are the (R)LWE secret dimension, the number of codewords with a Hamming weight of 1 and the number of database elements). Based on PIRCOR, we further present PIRCOR* by additionally introducing the *Rotation-self-Multiplication* operation, which achieves a **50%** reduction in rotation operations and a smaller query size when $\delta = 2$.

Building upon PIRCOR and PIRCOR*, we further propose their optimized variants, PIRCOR-op and PIRCOR*-op, to further reduce the online response time. Similar to YPIR that leverage preprocessing, PIRCOR-op and PIRCOR*-op allow all rotations and part of multiplications to be carried out in an offline stage before receiving the query. Additionally, we also design *FHE-operator acceleration with leveled optimization* and *implementation optimization of ciphertext rotation*.

For 8 KB element retrieval in an 8 GB database, PIRCOR achieves a $\mathbf{10.7}\times$ query size reduction compared to PIRANA. When benchmarked against YPIR, the improvements are even more striking: PIRCOR reduces the query size by $\mathbf{26.8}\times$ and accelerates query generation by a staggering $\mathbf{6,080}\times$. Notably, the enhanced PIRCOR* achieves a $\mathbf{53.6}\times$ reduction in query size compared to YPIR, while improving query generation time by an impressive $\mathbf{12,160}\times$.

CCS Concepts

• **Security and privacy → Privacy-preserving protocols**.

Keywords

Private information retrieval (PIR), FHE, Encoding technique

1 Introduction

Private information retrieval (PIR) enables a client to securely retrieve a specific element from a server-hosted database by its index while ensuring that the server remains oblivious to the index of the requested element. This concept was first introduced by Chor et al. [10] and has since served as a crucial building block in various applications, including metadata-hiding messaging [3, 29], private contact discovery [17], credential breach reporting [22, 32, 33], and private blocklist lookups [18], among others.

Based on the system architecture, the literature on PIR research can be roughly divided into two categories: Multi-Server PIR [11, 20, 35] and Single-Server PIR [16, 21, 23, 24, 27, 28, 30, 34]. Multi-server PIR schemes achieve better computational and communication efficiency by distributing queries across multiple non-colluding servers. However, their reliance on the assumption of non-collusion among servers is often impractical in real-world applications. In contrast, single-server PIR constructions eliminate the need for such an assumption but come at the cost of efficiency. Consequently, the primary research objective in the single-server PIR setting is to optimize concrete performance measures, including *the query size*, *the response size*[1], and *the server response time*[2], which is also the focus of this work.

In the following, we outline the current state-of-the-art single-server PIR solutions (to our knowledge) in two general categories.

Category I: Pre-processing-related PIR. Many designs within this category utilize the power of pre-processing, where either the server [12, 16, 21, 26, 28, 31, 34] or the client [1–3, 24, 27, 30] can perform some offline computation before the online query from the client.

- In server-side pre-processing constructions, a representative example is SimplePIR/DoublePIR [16], where the server computes a database-dependent "hint" during an offline phase and transmits it to the client for storage. It optimizes the server's throughput during the online phase, but incurs

*Both authors contributed equally to this research.
†Corresponding author.

[1] Some schemes substitute the response size with the rate, which is the ratio of the response size to the size of the retrieved record, and serves as a measure of the communication overhead from the server to the client.

[2] Some schemes substitute the server response time with the server throughput, which is the ratio of the database size to the server's response time, and measures how quickly the server can respond to a PIR query as a function of the database size.

high communication overhead due to the transmission of large hint data. Furthermore, schemes such as [15, 31, 34] can further improve server throughput by allowing the client to stream the entire database during the offline phase. Of course, these schemes are only practical when clients are able to perform a streaming download of the entire database. However, these hint-based approaches suffer from large offline download requirements and are unsuitable for dynamic databases.

- In client-side pre-processing constructions, many schemes, like FastPIR [1], SealPIR [3], Spiral [27], OnionPIR [30], enable the client to upload a "public key" to the server during an offline phase, thereby reducing both the query and response sizes. However, in these schemes, the server is required to store a (potentially large) public key for each client, which imposes significant storage overhead. Furthermore, SimplePIR/DoublePIR [16] highlights that the reuse of client-specific public keys may facilitate active attacks.
- Silent pre-processing. As stated in [21, 28], large offline communication costs in the above two constructions remain a major bottleneck in many of the most concretely efficient single-server PIR protocols. Consequently, a protocol with *silent pre-processing* that achieve good concrete efficiency with pre-processing, but *without* offline communication, is presented. A representative solution is HinelessPIR [21], which eliminates the client-side hint download required in SimplePIR [16] by enabling the client to upload an additional RLWE encryption of the LWE secret key along with a key-switching matrix, facilitating online bootstrapping [14]. However, this approach inevitably increases both the query size and the response time. The current state-of-the-art scheme, YPIR [28], overcomes the limitation of response time in HinelessPIR [21] by adopting the LWE-to-RLWE packing technique [7] instead of relying on bootstrapping. Since YPIR [28] is built upon DoublePIR [16], it is not well-suited for retrieving larger records (e.g., 256 bits), thereby limiting its applicability in many practical scenarios.

Category II: Coding-related PIR. Very recently, Mahdavi and Kerschbaum [25] introduce a new direction for PIR by using constant-weight codes to encode all indices and queries into codewords with the same Hamming weight, effectively reducing the query size without requiring any offline pre-processing. In the early stages of coding-related PIR development, the protocol proposed in [25], known as CwPIR, requires $(k-1)|\mathrm{DB}|$ ciphertext-ciphertext multiplications (where k is the Hamming weight and $|\mathrm{DB}|$ is the number of database elements), making it impractical for databases with a large number of elements. To reduce the response time, the most state-of-the-art PIRANA [23] integrates the constant-weight codes with the SIMD-based fully homomorphic encryption technique. Specifically, PIRANA [23] reduces the number of ciphertext-ciphertext multiplications from $(k-1)|\mathrm{DB}|$ in CwPIR [25] to $\left\lceil \frac{|\mathrm{DB}|}{N} \right\rceil (k-1)$, where N is the polynomial degree of polynomial rings.

Summary of two categories. To the best of our knowledge, pre-processing-based PIR schemes typically build upon underlying fully homomorphic encryption techniques by offloading part of the query-independent ciphertext computations to an offline pre-processing phase. This design helps optimize concrete performance metrics such as query size, response time, and response size. However, it inevitably introduces offline computation and storage overhead, and may even incur additional offline communication costs. More importantly, such overhead is often proportional to the size of the database, which limits the scalability and practicality of these schemes in large-scale database scenarios. Furthermore, any update to the database requires re-executing the entire offline pre-processing phase. As a result, the online performance gains brought by such offline processing become negligible, especially in scenarios where the database is continuously updated in real time [3].

On the other hand, Category II offers greater flexibility. It addresses the problem from the perspective of protocol design, using efficient coding techniques to directly reduce both the query size and response size without relying on any offline pre-processing. As a result, it eliminates the need for large-scale storage on either the client or server side. Of course, if further reduction in response time is desired, the pre-processing techniques used in Category I can be integrated as well.

Therefore, in this paper, we focus on the coding-related PIR. Unlike PIRANA [23] that adopts the constant-weight code, we construct a new coding method to further reduce both the query size and query generation time. Additionally, we explore how pre-processing techniques can be integrated with our coding method to further enhance computational performance.

1.1 Our results

This work makes substantial progress for single-server PIR in Category II, demonstrating contributions in the following aspects.

(1) Query-efficient PIRCOR: We construct the *Rotation-based Expanded Binary Code* ($\mathbf{REBC}(N, n, \alpha, \beta, \delta)$) to expand α N-dimensional codewords into β N-dimensional codes through *Rotation-Mutual-Multiplication* (RMM) across δ distinct codewords (i.e., $\boldsymbol{y}_l = \boldsymbol{x}_{i_0} \circ \boldsymbol{x}_{i_1}^{j_1} \circ \cdots \circ \boldsymbol{x}_{i_{\delta-1}}^{j_{\delta-1}}$). Based on $\mathbf{REBC}(\cdot)$, we design the PIRCOR, a communication-optimal hintless single-server PIR scheme without pre-processing technique. In PIRCOR, we encode the query index into α codewords, where δ of them have a Hamming weight of 1 and the remaining $\alpha - \delta$ are all-zero codewords. During the response phase, the server performs the RMM operation to expand these codewords into β expanded codewords, used for exactly retrieving the target element. This encoding technique enables a reduction in both the query size and time for single-query PIR by a factor of $O\left(N^{\frac{\delta-1}{\delta}}\right)$ compared to PIRANA [23]. For example, if $N = 2^{12}$ and the number of database elements is $|\mathrm{DB}| = 2^{24}$, the query size of our PIRCOR scheme is only 2 RLWE ciphertexts, whereas PIRANA [23] requires sending 92 RLWE ciphertexts.

(2) Computational-efficient PIRCOR*: Building on PIRCOR, we propose PIRCOR* to further reduce the cost of ciphertext rotation operations in the case where $\delta = 2$. Specifically, we achieve

[3] Since YPIR [28] eliminates offline communication, it claims the capability to support real-time database updates and presents experimental results on private Signed Certificate Timestamp (SCT) auditing in Certificate Transparency [16]. However, the scheme and its evaluation do not take into account the offline computation and storage overhead, which scale with the size of the database. As a result, the conclusions are drawn solely from the perspective of offline communication, lacking a comprehensive assessment of the total system overhead.

the further expansion by additionally considering the *Rotation-Self-Multiplication* (RSM) (i.e., $\boldsymbol{y}_l = \boldsymbol{x}_i \circ \boldsymbol{x}_i^j$ for $i \in [\alpha]$). Compared with PIRCOR, PIRCOR* reduces the number of ciphertext rotations by 50%. Notably, PIRCOR* can even shrink both the query size and query generation time.

(3) Implementation optimization: We propose the optimized variants, PIRCOR-op and PIRCOR*-op, by integrating some additional techniques into our PIRCOR and PIRCOR* to further improve concrete performance.

- *Pre-processing integration*: Similar to many state-of-the-art PIR schemes [16, 28] that leverage pre-processing, we propose to shift all rotation operation and query-independent intensive computation into an offline part to split offline and online costs by a common reference string (CRS) model. This integration halves the query size and reduce the response time up to 50%.
- *FHE Operator Acceleration with Leveled Optimization*: The underlying FHE scheme employs a leveled computation architecture where higher-level operations incur greater latency. To optimize this hierarchy: 1) We strategically prioritize lightweight plaintext-ciphertext multiplications to higher levels while reserving lower levels for computationally heavy ciphertext-ciphertext multiplications to reduce the overall computational cost, and 2) Introduce adaptive modulus switching before each homomorphic operation to execute it (especially rotation) at the minimal permissible level without compromising algorithm correctness. This dual optimization approach is seamlessly applicable to PIRANA's framework [23]. Note that this optimization technique is particularly suitable for scenarios with small element sizes.
- *Implementation optimization of ciphertext rotation*: In Single Instruction Multiple Data (SIMD)-based FHE, rotations are occur along multiple independent dimensions. Thus, a naive rotation via Galois automorphism cannot achieve our code extension method desired cyclic left shift across all slots. To address this issue, we propose alternative solutions by optimizing from two aspects: the computation-efficiency design and the query-size-efficiency solution. If we prioritize minimizing the query size, we can opt for a query-size-efficient approach that trades computation for communication efficiency. Otherwise, we can choose a computation-efficiency strategy to reduce computational overhead.

(4) Performance: We compare our four schemes, i.e, PIRCOR, PIRCOR*, PIRCOR-op and PIRCOR*-op, with state-of-the-art PIR solutions, such as PIRANA [23] (a coding-based PIR) and YPIR [28] (a preprocessing-based PIR), in terms of the number of the database elements (i.e., |DB|) and the element size (i.e., |db|) to evaluate their concrete performance.

- *Query size and time comparison*: As the database size increases, our four schemes significantly outperform than PIRANA in terms of the query size. Notably, the advantages of our designs become increasingly prominent as |DB| scales. For instance, the query size of PIRANA is 60× larger than that of PIRCOR*-op. For large element sizes, our schemes achieve substantial reductions in query size compared to YPIR, with PIRCOR*-op demonstrating improvements by orders of magnitude. Moreover, the advantage becomes increasingly prominent as |DB| grows. For an 8GB database ($2^{20} \times 8$KB), our PIRCOR*-op outperforms YPIR by **99**%. Our four schemes offer significant improvements in query generation time compared to both PIRANA and YPIR. For an 8GB database ($2^{20}\times$8KB), our PIRCOR* (and PIRCOR*-op) achieves a query generation time of **5 ms**, outperforming PIRANA and YPIR by factors of **9.4**× and **12,160**×, respectively.
- *Response size and time comparison*: In the original versions of PIRCOR and PIRCOR*, the response time is higher than that of PIRANA. However, in our optimized versions, the response time is reduced by up to 50% compared to PIRANA. The response sizes of our four schemes are identical to that of PIRANA, while larger than that of YPIR.

2 Preliminaries

Notations. We use $\mathbb{Z}$ and $\mathbb{R}$ to denote the set of integers and real numbers, respectively. For a positive $x \in \mathbb{Z}$, let $[x]$ be the set of integers $\{0, 1, \ldots, x-1\}$. For any integers $x, y \in \mathbb{N}$ such that $x < y$, let $[x, y]$ and $[x, y)$ be the set of integers $\{x, x+1 \ldots, y\}$ and $\{x, x+1, \ldots, y-1\}$, respectively. For $x \in \mathbb{R}$, $\lceil x \rceil$ and $\lfloor x \rfloor$ denote the ceiling and floor, respectively.

In this paper, a vector is denoted by a bold letter, e.g., $\boldsymbol{x}$, where the i-th element is denoted as $\boldsymbol{x}[i]$. All vectors have dimension N with indices starting from 0 and ending at $N-1$, e.g., $\boldsymbol{x} = (\boldsymbol{x}[0], \boldsymbol{x}[1], \ldots, \boldsymbol{x}[N-1])$. We use $\widetilde{\boldsymbol{x}}$ to denote the SIMD ciphertext obtained by encrypting the plaintext vector $\boldsymbol{x}$. Note that the symbols i, j, k, t, d and l represent arbitrary variables without any specific meaning.

2.1 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is a groundbreaking cryptographic technique that enables arbitrary computations to be performed directly on encrypted data without requiring decryption. This property is highly valuable in privacy-preserving computing applications, such as secure PIR and privacy-preserving machine learning.

Polynomial Rings. In general, most FHE schemes operate on polynomial rings [4, 6, 9], usually denoted $R = \mathbb{Z}[x]/(x^N+1)$, where N is a power of 2. Specifically, any message should be encoded into the plaintext space, represented as a polynomial in the quotient ring $R_p := R/pR = \mathbb{Z}_p[x]/(x^N+1)$, where p is a positive integer known as the plaintext modulus. The plaintext polynomial is then encrypted into ciphertext polynomial in the quotient ring $R_q := \mathbb{Z}_q[x]/(x^N+1)$, where q is the ciphertext modulus.

Single Instruction Multiple Data (SIMD) is a parallel computing technique that enables the simultaneous processing of multiple elements under a single instruction. In most RLWE-based cryptosystems [8, 9, 13], SIMD technology has been adopted to significantly improve computational efficiency, especially when handling large-scale data. More specifically, for an N-dimensional vector of elements, e.g., $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1}) \in \mathbb{Z}_p^N$, the SIMD-based FHE encodes it by constructing a polynomial $y(X) = \sum_{i\in[N]} y_i X^{i-1}$ where $x_i = y(\zeta_j)$, $\zeta_j := \zeta^{3^j} \mod p$, and ζ is the $2N$-th primitive

root of unity of p. Then one can encrypt it into a single SIMD ciphertext $\widetilde{\boldsymbol{x}} \in R_q^2$, allowing N encrypted elements to be processed in parallel with single FHE operation.

The SIMD-based FHE scheme used in this paper are summarized below.

- $\mathrm{pp} = (N, p, q, \mathcal{X}) \leftarrow \mathbf{Setup}(\lambda)$. The setup algorithm takes the secure parameter λ as input and outputs public parameters, which includes the polynomial degree N, the plaintext modulus p, the ciphertext modulus q, and an error distribution $\mathcal{X}$ over R_q.
- $(\mathbf{sk}, \mathbf{evk}) \leftarrow \mathbf{KeyGen}(\mathcal{X})$. The key generation algorithm takes the public parameter $\mathcal{X}$ as input, samples the secret key $\mathbf{sk} \in \mathcal{X}$ and generates the FHE evaluation key $\mathbf{evk}$. $\mathbf{evk}$ can be seen as a variant of encrypted $\mathbf{sk}$ that is publicly available and serves as input to the follwing evaluation algorithms.
- $\widetilde{\boldsymbol{x}} = (\widetilde{\boldsymbol{x}}[1], \widetilde{\boldsymbol{x}}[2]) \leftarrow \mathbf{SIMDEnc}(\mathbf{sk}, \boldsymbol{x})$. The encryption algorithm encrypts the N-dimensional plaintext vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1})$ using the secret key $\mathbf{sk}$ to generate the SIMD ciphertext $\widetilde{\boldsymbol{x}} \in R_q^2$.
- $\boldsymbol{x} \leftarrow \mathbf{SIMDDec}(\mathbf{sk}, \widetilde{\boldsymbol{x}})$. The decryption algorithm decrypts the SIMD ciphertext $\widetilde{\boldsymbol{x}}$ using the secret key $\mathbf{sk}$ to recover the corresponding plaintext vector $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1})$.
- $\widetilde{\boldsymbol{z}} \leftarrow \mathbf{SIMDAdd}(\widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}})$. The ciphertext addition algorithm takes two SIMD ciphertexts, $\widetilde{\boldsymbol{x}}$ and $\widetilde{\boldsymbol{y}}$, as inputs and outputs the SIMD ciphertext $\widetilde{\boldsymbol{z}}$, which is the encryption of the plaintext vector $\boldsymbol{x} + \boldsymbol{y} = (x_0 + y_0, x_1 + y_1, \ldots, x_{N-1} + y_{N+1})$.
- $\widetilde{\boldsymbol{z}} \leftarrow \mathbf{SIMDPmul}(\widetilde{\boldsymbol{x}}, \boldsymbol{y})$. The plaintext-ciphertext multiplication algorithm takes an SIMD ciphertext $\widetilde{\boldsymbol{x}}$ and a plaintext vector $\boldsymbol{y} = (y_0, y_1, \ldots, y_{N-1})$ as inputs, and outputs an SIMD ciphertext $\widetilde{\boldsymbol{z}}$, which is the encryption of the plaintext vector $(x_0 y_0, x_1 y_1, \ldots, x_{N-1} y_{N-1})$.
- $\widetilde{\boldsymbol{z}} \leftarrow \mathbf{SIMDMul}(\mathbf{evk}, \widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}})$. The ciphertext multiplication algorithm takes the evaluation key $\mathbf{evk}$, two SIMD ciphertexts $\widetilde{\boldsymbol{x}}$ and $\widetilde{\boldsymbol{y}}$ as inputs, and outputs an SIMD ciphertext $\widetilde{\boldsymbol{z}}$, which is the encryption of the plaintext vector $(x_0 y_0, \ldots, x_{N-1} y_{N-1})$.
- $\widetilde{\boldsymbol{x}}^j \leftarrow \mathbf{SIMDRotate}(\mathbf{evk}, \widetilde{\boldsymbol{x}}, j)$. The SIMD rotation algorithm takes the evaluation key $\mathbf{evk}$, an SIMD ciphertext $\widetilde{\boldsymbol{x}}$ and a rotation index $j \in [N]$ as inputs, and outputs an SIMD ciphertext $\widetilde{\boldsymbol{x}}^j$, which is the encryption of the plaintext vector $(x_{(0+j)\%N}, x_{(1+j)\%N}, \ldots, x_{(N-1+j)\%N})$. Note that $j = 0$ means no rotation.

Modulus switching. The core operation of modulus switching involves reducing the modulus of the ciphertext to a smaller value while cutting the least significant bit noise, ensuring decryption correctness while controlling noise level [5, 27]. Specifically, Modulus switching takes the ciphertext modulo q and rescales it to the ciphertext modulo q_1 where $q_1 < q$, as formalized below.

- $\widetilde{\boldsymbol{x}}_{q,q_1} \leftarrow \mathbf{ModSwitch}(\widetilde{\boldsymbol{x}}, q_1)$: For integers $q > q_1$ and on input the ciphertext $\widetilde{\boldsymbol{x}} \in R_q^2$, output $\widetilde{\boldsymbol{x}}_{q,q_1} \in R_{q_1}^2$.

2.2 Single-Server Private Information Retrieval

In this section, we recall the formal syntax of a single-server PIR protocol [19]. Similar to [16, 23, 28], we consider a model where an initial setup phase outputs a set of public parameters along with predefined database-dependent parameters.

Definition 1. A single-server PIR scheme, denoted as $\Pi_{\mathrm{PIR}} = (\mathbf{SysSet}, \mathbf{Query}, \mathbf{Answer}, \mathbf{Extract})$, consists of following algorithms:

- $\mathbf{SysSet}(\lambda, \mathrm{DB}) \rightarrow (\mathrm{pp}, \mathrm{dbp})$: On input the security parameter λ related to the FHE and a database $\mathrm{DB} = \{\mathrm{db}_0, \ldots, \mathrm{db}_{|\mathrm{DB}|-1}\}$, the server performs the system setup algorithm to output a set of public parameters pp and database-dependent parameters dbp.
- $\mathbf{Query}(\mathrm{pp}, \mathrm{idx}) \rightarrow (\mathrm{query}, \mathrm{sk})$: On input the public parameters pp and an index idx of the element to be retrieved in DB, the client performs the query algorithm to output the query query and a secret key sk.
- $\mathbf{Answer}(\mathrm{query}, \mathrm{dbp}) \rightarrow \mathrm{resp}$: On input the query query and database-dependent parameters dbp, the server performs the answer algorithm to output a response resp.
- $\mathbf{Extract}(\mathrm{resp}, \mathrm{sk}) \rightarrow \mathrm{db}_{\mathrm{idx}}$: On input a response resp and the client's secret key sk, the client performs the extract algorithm to output the element $\mathrm{db}_{\mathrm{idx}}$.

The formal correctness and security definitions for PIR can be referred to [28].

3 Our Scheme

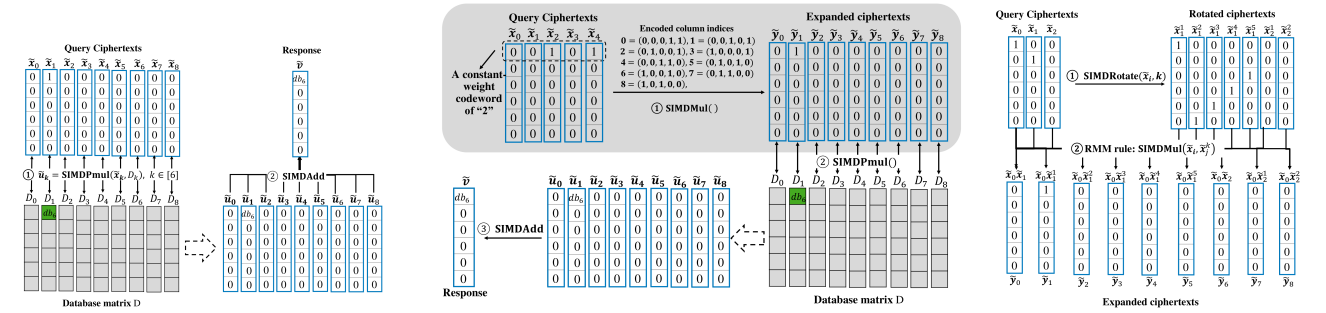
3.1 Overview of PIRCOR

Intuition. Similarly to most single-server PIR scheme, we arrange the database $\mathrm{DB} = \{db_0, db_1, \ldots, db_{|\mathrm{DB}|-1}\}$ into a database matrix D of N rows and $\beta := \lceil |\mathrm{DB}|/N \rceil$ columns, where $|\mathrm{DB}|$ is the size of DB. In this arrangement, the element with index idx is placed in row $r := \mathrm{idx}\%N$ and column $c := \lfloor \mathrm{idx}/N \rfloor$. Note that N is the degree of the polynomial ring. Under this setting, each column of D can be directly utilized for SIMD-based homomorphic operations like $\mathbf{SIMDEnc}(\cdot)$ and $\mathbf{SIMDPmul}(\cdot)$ (see Section 2.1).

3.1.1 Naive method. To retrieve the element db_{idx}, as illustrated in Fig. 1a, the naive method is that the client directly sends β SIMD ciphertexts, where only the r-th slot of the c-th ciphertext is 1 and all other slots are 0s. The server simply performs plaintext-ciphertext multiplication $\mathbf{SIMDPmul}(\cdot)$ between the i-th column of the database matrix D and the i-th SIMD ciphertex for $i \in [\beta]$, then sums the resulting products to obtain the response ciphertext. Finally, db_{idx} is in the r-th slot of the response ciphertext. However, the query size in this method is β SIMD ciphertexts.

3.1.2 PIRANA [23]. To reduce the query size, the state-of-the-art PIRANA introduces a constant-weight codeword $\mathrm{CW}(\alpha, \delta)$ with the code length α and Hamming weight δ to encode the column indices $\{0, 1, \ldots, \beta - 1\}$ as β constant-weight codewords. As shown in Fig. 1b, the client only sends α SIMD ciphertexts to the server. Based on β constant-weight codewords, the server performs additional β ciphertext multiplication $\mathbf{SIMDMul}(\cdot)$ to expand α SIMD ciphertexts to β SIMD ciphertexts, which are identical to those sent by the client in the aforementioned naive method. In PIRANA, the query size is reduced from β to $O(\sqrt[\delta]{\delta!\beta} + \delta)$, while the computational costs of the server increase by $(\delta - 1)\beta$ SIMD ciphertext multiplications.

3.1.3 Our PIRCOR. The starting point for this work is the PIRANA [23] based on the coding method. Unlike PIRANA [23], we construct a *Rotation-based Expanded Binary Code* (see Definition 2) to further reduce the query size. Fig. 1c visualizes this process.



(a) Naive method **(b) PIRANA [23]** **(c) Our PIRCOR**

Figure 1: Toy example showing workflows of the Naive method, PIRANA [23], and PIRCOR with $N = 6$, $|\mathrm{DB}| = 54$, and $\delta = 2$. PIRCOR reduces query size by modifying PIRANA's gray-shaded parts. Query sizes: Naive = 9, PIRANA = 5, PIRCOR = 3.

Definition 2 (Rotation-based expanded binary code)**.** Let N, α, δ and β be the codeword length, the number of primary codewords, the number of primary codewords with a Hamming weight of 1, and the number of expanded codewords, respectively. A Rotation-Based Expanded Binary code (REBC), denoted as $\mathrm{REBC}(N, \alpha, \beta, \delta)$, expands α N-length primary codewords $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\alpha-1}\}$ into β N-length expanded codewords $\{\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{\beta-1}\}$. Among these β expanded codewords, only one has a Hamming weight of 1, while all others are entirely zero.

The $\mathrm{REBC}(N, \alpha, \beta, \delta)$ includes the following two operations:

(1) **Codeword expansion**: This operation expands α primary codewords into β expanded codewords through *Rotation-Mutual-Multiplication* (RMM) across distinct primary codewords. The `CodeExpand(·)` is detailed in Algorithm 1.
(2) **Primary codeword generation**: Generate α N-dimensional primary codewords $(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{\alpha-1})$, where δ codewords have a Hamming weight of 1, while the remaining $\alpha - \delta$ are all-zero codewords. Note that the indices of δ codewords with Hamming weight 1 and the corresponding positions of the 1 within them are determined by the index of the unique expanded codeword with Hamming weight 1 (denoted as c) and its position of the 1 (denoted as r) among β expanded codeword. The primary codeword generation `CodeGen(·)` is given in Algorithm 2.

Remark 1 (Explanation about `CodeExpand(·)`)**.** Here, we provide explanations for several particular configurations.

- As is well known, an N-dimensional codeword can be circularly rotated up to $N-1$ positions, resulting in N distinct rotations including the original.
- For each combination of δ N-dimensional codewords, there are N^δ possible element combinations. Note that $\boldsymbol{x}_{i_0} \circ \boldsymbol{x}_{i_1}^{j_1 - j_0} \circ \boldsymbol{x}_{i_2}^{j_2 - j_0} \circ \cdots \circ \boldsymbol{x}_{i_{\delta-1}}^{j_{\delta-1} - j_0}$ and $\boldsymbol{x}_{i_0}^{j_0} \circ \boldsymbol{x}_{i_1}^{j_1} \circ \boldsymbol{x}_{i_2}^{j_2} \circ \cdots \circ \boldsymbol{x}_{i_{\delta-1}}^{j_{\delta-1}}$ generate the same N element combinations for all $0 \le j_0 < N$. Thus, no rotation is applied to $\boldsymbol{x}_{i_0}$ in Lines 2–4, and $j_0 = 0$ is always set in Line 8.
- According to Line 6, randomly selecting δ distinct codewords from the set $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\alpha-1}\}$ yields a total of C_α^δ unique combinations. Throughout this paper, these combinations are then reordered in lexicographic (i.e., increasing) order based on their indices $(i_0, \ldots, i_{\delta-1})$ such that $0 \le i_0 < i_1 < \cdots < i_{\delta-1} \le \alpha - 1$. Specifically, we define $(i_0, \ldots, i_{\delta-1}) < (j_0, \ldots, j_{\delta-1})$ if there exists an index l where $i_l < j_l$ and $i_s = j_s$ for all $0 \le s < l$. These C_α^δ combinations are then sequentially indexed by i, where $i = (i_0, i_1, \ldots, i_{\delta-1})$.

Based on Remark 1, the parameters α and β in $\mathrm{REBC}(N, \alpha, \beta, \delta)$ satisfy:

$$\beta = C_\alpha^\delta \cdot N^{\delta-1}. \tag{1}$$

Given a fixed β, the number of primary codewords is $O(\sqrt[\delta]{\delta!\beta/N^{\delta-1}})$. *Consequently, our PIRCOR achieve a further reduction by a factor of* $O\left(N^{\frac{\delta-1}{\delta}}\right)$ *compared to the PIRANA [23].*

Algorithm 1: Codeword expansion function

```
1  Function CodeExpand({x_0, x_1, ..., x_{α-1}})
2      for i ← 1 to α − 1 do
3          for j ← 1 to N − 1 do
4              x_i^j = (x_i[(0 + j)%N], ..., x_i[(N − 1 + j)%N]) ;
5      Set l = 0 ;
6      for any 0 ≤ i_0 < i_1 < ··· < i_{δ−1} ≤ α − 1 do
7          for any 0 ≤ j_1, j_2, ..., j_{δ−1} ≤ N − 1 do
8              y_l = x_{i_0} ∘ x_{i_1}^{j_1} ∘ x_{i_2}^{j_2} ∘ ··· ∘ x_{i_{δ−1}}^{j_{δ−1}} ;
9              l = l + 1 ;
10     return β expanded codes {y_0, y_1, ..., y_{β−1}}
```

Remark 2 (Explanation about Line 7-11 in `CodeGen(·)`)**.** Here we describe how to determine the original position of the 1 in the N-dimensional non-zero primary codewords $\{\boldsymbol{x}_{\phi_0}, \boldsymbol{x}_{\phi_1}, \ldots, \boldsymbol{x}_{\phi_{\delta-1}}\}$. Given the desired target position r, it follows from Line 8 of Algorithm 1 that $y[r] = \boldsymbol{x}_{\phi_0}[r] \circ \ldots \circ \boldsymbol{x}_{\phi_{\delta-1}}[r + j_{\delta-1}] = 1$, which results in Line 11 of Algorithm 2.

Algorithm 2: Primary codewords generation

1 **Function** `CodeGen(`r, c`)`
2 **for** $l \in [\alpha]$ **do**
3 Set $\boldsymbol{x}_l = \mathbf{0}^N$ ▷ N-dimensional zero vector;
4 Find unique $\phi \in \left[C_\alpha^\delta\right]$ and $\varphi \in [N^{\delta-1}]$ such that
$$c = \phi \cdot N^{\delta-1} + \varphi$$
5 Retrieve the indices of the δ non-zero codewords in the ϕ-th indexed codeword combination in ascending order, i.e., $\phi = (\phi_0, \phi_1, \ldots, \phi_{\delta-1})$ where $\phi_0 < \phi_1 < \cdots < \phi_{\delta-1} \leq \alpha - 1$;
6 Set $k = \delta$ and $t_k = \varphi$;
7 $\boldsymbol{x}_{\phi_0}[r] = 1$;
8 **for** $l = \phi_{\delta-1}, \phi_{\delta-2}, \ldots, \phi_1$ **do**
9 $j_{k-1} = t_k \bmod N$;
10 $t_{k-1} = \frac{t_k - j_{k-1}}{N}$;
11 Set $\boldsymbol{x}_l[(r + j_{k-1})\%N] = 1$;
12 $k = k - 1$;
13 **return** $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\alpha-1}\}$;

3.2 The PIRCOR scheme

In this section, we detail how to utilize $\mathrm{REBC}(N, \alpha, \beta, \delta)$ to implement PIR effectively. Similar to Definition 1, our PIRCOR scheme, represented as **PIRCOR** = (**SysSet**, **Query**, **Answer**, **Extract**), is defined below.

3.2.1 **SysSet**$(\lambda, \mathrm{DB}) \to (\mathrm{pp}, \mathrm{dbp})$. On input the security parameter λ and a database $\mathrm{DB} = \{\mathrm{db}_0, \mathrm{db}_1, \ldots, \mathrm{db}_{|\mathrm{DB}|-1}\}$, the server proceeds the one-time setup algorithm as follows:

(1) Generate RLWE-related public parameters $\mathrm{pp} = (N, p, q, \mathcal{X})$ by executing **Setup**(λ).
(2) For the database DB, arrange its |DB| elements into a database matrix D of N rows and $\beta := \lceil |\mathrm{DB}|/N \rceil$ columns. For simplicity, we assume that N divides |DB|, i.e., $\beta = |\mathrm{DB}|/N$. The element with index idx, i.e., $\mathrm{db}_{\mathrm{idx}}$, is placed in row $r := \mathrm{idx}\%N$ and column $c := \lfloor \mathrm{idx}/N \rfloor$: $\mathrm{D} = \{D_0, \ldots, D_{\beta-1}\}$, where $D_i = (\mathrm{db}_{iN}, \mathrm{db}_{iN+1}, \ldots, \mathrm{db}_{(i+1)N-1})$ is the column vector for $i \in [\beta]$.

The system setup algorithm outputs the RLWE-related public parameters $\mathrm{pp} = (N, p, q, \mathcal{X})$ and the database-related parameters $\mathrm{dbp} = (\beta, \mathrm{D})$.

3.2.2 **Query**$(\mathrm{pp}, \beta, \mathrm{idx}) \to (\mathrm{query}, \mathbf{evk}, \mathbf{sk})$. On input the public parameter $\mathrm{pp} = (N, p, q, \mathcal{X})$, the number of database columns β and an index $\mathrm{idx} \in [|\mathrm{DB}|]$ of the element to be retrieved in the database (i.e., $\mathrm{db}_{\mathrm{idx}}$), the client executes the query algorithm as outlined in Algorithm 3.

(1) Based on N and β, determine the query size α using Eq. (1). It is worth noting that if the database size |DB| remains constant, the value of α does not change, and therefore it only needs to be computed once (Line 1).
(2) Generate the secret key **sk** and the evaluation key **evk** by implementing **KeyGen**$(\mathcal{X})$ (Line 2).
(3) Determine the row index $r \in [N]$ and column index $c \in [\beta]$ in D based on the query index $\mathrm{idx} \in [|\mathrm{DB}|]$ (Lines 3-4).
(4) Generate α codewords $\{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{\alpha-1}\}$ from `CodeGen(·)`, (Line 5).
(5) For each $l \in [\alpha]$, perform the encryption algorithm $\widetilde{\boldsymbol{x}}_l \leftarrow$ **SIMDEnc**$(\mathbf{sk}, \boldsymbol{x}_l)$ to obtain α SIMD ciphertexts for querying (Lines 6-7).

Finally, output the query $\mathrm{query} = (\widetilde{\boldsymbol{x}}_0, \widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_{\alpha-1})$ and the query key $(\mathbf{evk}, \mathbf{sk})$. The client sends query and **evk** to the server and keeps **sk** secret.

Algorithm 3: Query encoding

Input: The query index idx, the public parameter pp and the number of database columns β
Output: α query SIMD ciphertexts $\{\widetilde{\boldsymbol{x}}_0, \widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_{\alpha-1}\}$ and the query key $(\mathbf{evk}, \mathbf{sk})$

1 Based on N and β, determine the query size α using Eq. (1) ;
2 $(\mathbf{sk}, \mathbf{evk}) \leftarrow$ **KeyGen**$(\mathcal{X})$;
3 Compute the row index $r := \mathrm{idx}\%N$;
4 Compute the column index $c := \lfloor \mathrm{idx}/N \rfloor$;
5 Run $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\alpha-1}\} \leftarrow$ `CodeGen(`r, c`)` ;
6 **for** $l \leftarrow 0$ **to** $\alpha - 1$ **do**
7 $\widetilde{\boldsymbol{x}}_l \leftarrow$ **SIMDEnc**$(\mathbf{sk}, \boldsymbol{x}_l)$
8 **return** $\{\widetilde{\boldsymbol{x}}_0, \widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_{\alpha-1}\}$ and $(\mathbf{evk}, \mathbf{sk})$;

3.2.3 **Answer**$(\mathrm{query}, \mathbf{evk}, \mathrm{D}) \to \mathrm{resp}$. On input the query $\mathrm{query} = (\widetilde{\boldsymbol{x}}_0, \ldots, \widetilde{\boldsymbol{x}}_{\alpha-1})$, the evaluation key **evk**, and the database column vectors $\mathrm{D} = \{D_0, \ldots, D_{\beta-1}\}$, the server proceeds Algorithm 4 to output the response $\mathrm{resp} = \widetilde{\boldsymbol{v}}$, which is then sent to the client.

Algorithm 4: Answer

Input: query $= (\widetilde{\boldsymbol{x}}_0, \widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_{\alpha-1})$, the evaluation key **evk**, and the database matrix $\mathrm{D} = \{D_0, \ldots, D_{\beta-1}\}$
Output: The response $\widetilde{\boldsymbol{v}}$, where the value of $r = [\mathrm{idx}\%N]$-th slot is $\mathrm{db}_{\mathrm{idx}}$ and the values of all other slots are 0s

1 Run `CodeExpand(`$\{\widetilde{\boldsymbol{x}}_0, \widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_{\alpha-1}\}$`)` in Algorithm 1, and output $(\widetilde{\boldsymbol{y}}_0, \widetilde{\boldsymbol{y}}_1, \ldots, \widetilde{\boldsymbol{y}}_{\beta-1})$;

`/* The codewords rotation in line 4 of Algorithm 1 is revised as:`
$$\widetilde{\boldsymbol{x}}_i^j \leftarrow \mathbf{SIMDRotate}(\mathbf{evk}, \widetilde{\boldsymbol{x}}_i, j)$$
`*/`

`/* The codes expansion in line 11 is revised as:`
$$\widetilde{\boldsymbol{y}}_l \leftarrow \mathbf{SIMDMul}(\mathbf{evk}, \widetilde{\boldsymbol{x}}_{i_0}, \widetilde{\boldsymbol{x}}_{i_1}^{j_1}, \ldots, \widetilde{\boldsymbol{x}}_{i_{\delta-1}}^{j_{\delta-1}})$$
`*/`

2 **for** $l \leftarrow 0$ **to** $\beta - 1$ **do**
3 $\widetilde{\boldsymbol{u}}_l \leftarrow$ **SIMDPmul** $(\widetilde{\boldsymbol{y}}_l, D_l)$;
4 $\widetilde{\boldsymbol{v}} = \mathbf{SIMDAdd}\left(\widetilde{\boldsymbol{u}}_0, \widetilde{\boldsymbol{u}}_1, \ldots, \widetilde{\boldsymbol{u}}_{\beta-1}\right)$;
5 **return** $\widetilde{\boldsymbol{v}}$;

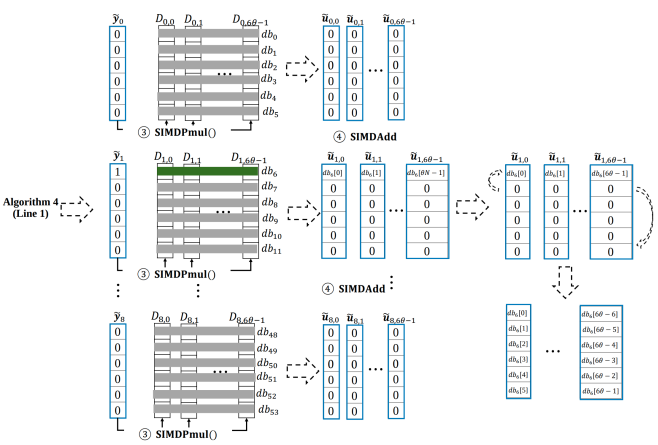


Figure 2: Workflow of PIRCOR for large-sized elements.

3.2.4 **Extract**$(\text{resp}, \mathbf{sk}) \to \text{db}_{\text{idx}}$. On input the response resp $= \widetilde{\boldsymbol{v}}$ and the client's secret key $\mathbf{sk}$, the client performs the extract algorithm as follows:

(1) Decrypt resp $= \widetilde{\boldsymbol{v}}$ with $\mathbf{sk}$: $\boldsymbol{v} \leftarrow \mathbf{SIMDDec}\,(\text{sk}, \widetilde{\boldsymbol{v}})$.
(2) Extract the value at the r-th position in the vector $\boldsymbol{v}$ as output.

3.3 Discussion about the size of element

In the design of Section 3.2, we only consider the element size (i.e., $|\text{db}_k|$ for $k \in [|\text{DB}|]$) to be smaller than the plaintext size (i.e., $|\text{db}_k| < |p|$), ensuring that each slot can encode exactly one element. Therefore, in this section, we discuss the scenario where the element size exceeds the plaintext size, i.e., $|\text{db}_k| > |p|$. As described in [23], the returned ciphertext $\widetilde{\boldsymbol{v}}$ in Algorithm 4 contains only a single non-empty slot. Therefore, we can fully utilize the remaining empty slots to retrieve a larger element.

As illustrated in Fig. 2, if each database element must be encoded into θ plaintext polynomials, i.e., the element size is $|\text{db}_i| = \theta \cdot N \cdot |p|$, we first divide each element into $\theta \cdot N$ smaller blocks. Then, for each expanded ciphertext $\widetilde{\boldsymbol{y}}_l$, the server performs $\mathbf{SIMDPmul}(\widetilde{\boldsymbol{y}}_l, D_{l,i})$ for each $i \in [\theta \cdot N]$ to generate $\theta \cdot N$ intermediate ciphertexts. Each ciphertext contains one non-empty slot, and the combination of these non-empty slots reconstructs the target element. Using $\mathbf{SIMDRotate}(\cdot)$, the server rotates the $\theta \cdot N$ ciphertexts to interlace their non-empty slots. Finally, all rotated ciphertexts are summed together, resulting in θ response ciphertexts that collectively encode the complete element. The differences are shown as follows.

- In $\mathbf{SysSet}(\lambda, \text{DB})$, the only difference is the initialization of database matrix. Specifically, split each db_i into $\text{db}_i = \text{db}_i[0] \| \cdots \| \text{db}_i[\theta N - 1]$ and initialize the database matrix $\text{D}^* = (D_{0,0}, \ldots, D_{\beta-1,\theta N-1})$, where $D_{i,j} = (\text{db}_{iN}[j], \ldots, \text{db}_{(i+1)N-1}[j])$ for $i \in [\beta]$ and $j \in [\theta N]$.
- In $\mathbf{Answer}(\text{query}, \mathbf{evk}, \text{D}^*)$, Algorithm 4 is revised to Algorithm 9, which is outlined in Appendix A.

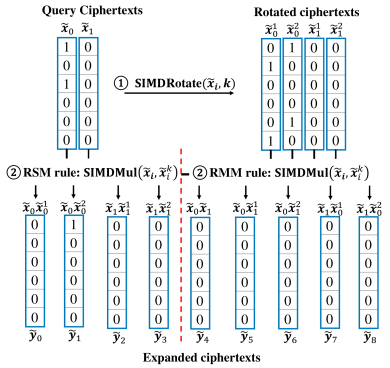


Figure 3: PIRCOR* scheme (using the same example as in Fig. 1), where the query size is further reduced from 3 in PIRCOR to 2.

4 Improvement of PIRCOR

In this section, we design PIRCOR* to reduce the response time of PIRCOR.

Intuition: As illustrated in Algorithm 1, the current code expansion strategy merely leverages *Rotation-Mutual-Multiplication* (RMM) across distinct primary codewords, i.e., $\boldsymbol{y}_l = \boldsymbol{x}_{i_0} \circ \boldsymbol{x}_{i_1}^{j_1} \circ \cdots \circ \boldsymbol{x}_{i_{\delta-1}}^{j_{\delta-1}}$. Actually, further expansion can be achieved by additionally considering the *Rotation-Self-Multiplication* (RSM), i.e., multiplying each primary codewords with its own $\delta - 1$ rotated forms, formulated as $\boldsymbol{y}_l = \boldsymbol{x}_i \circ \boldsymbol{x}_i^{j_1} \circ \cdots \circ \boldsymbol{x}_i^{j_{\delta-1}}$ for $i \in [\alpha]$. The integration of RMM and RSM effectively reduces the rotation cost, and can even shrink the query size and the corresponding generation time, particularly in the case of $\delta = 2$ and $\alpha < N - 1$. For clarity, we present our PIRCOR* construction under the case of $\delta = 2$, without loss of generality. Fig. 3 visualizes this process.

4.1 The PIRCOR* Scheme

Similar to PIRCOR in Section 3.2, the PIRCOR* also consists of the four algorithms, where **SysSet** and **Extract** are identical to those in PIRCOR. The key distinction is that, with the incorporation of RSM, the `CodeGen(·)` and `CodeExpand(·)` functions employed in the **Query** and **Answer** algorithms of PIRCOR must be replaced with `CodeGenImp(·)` and `CodeExpandImp(·)`, respectively.

In what follows, We begin by introducing `CodeGenImp(·)` and `CodeExpandImp(·)`, followed by a detailed discussion of how the **Query** and **Answer** algorithms in PIRCOR* differ from those in the original PIRCOR.

4.1.1 Codeword expansion function. Given that PIRCOR* enhances the codeword expansion of PIRCOR by additionally considering the RSM operation, the corresponding modifications have been integrated into the codeword expansion algorithm, as detailed in Algorithm 5.

Remark 3 (Explanation about `CodeExpandImp(·)`)**.** Here we explain the maximum rotation offset (i.e., $0 \le j \le N/2 - 1$) as well as the expansion rules of RSM and RMM (Lines 8, 12, and 15).

- Based on the two facts: for any two codewords $\boldsymbol{x}_{i_0}$ and $\boldsymbol{x}_{i_1}$ where $i_0, i_1 \in [\alpha]$, $\boldsymbol{x}_{i_0} \circ \boldsymbol{x}_{i_1}^{j}$ and $\boldsymbol{x}_{i_1} \circ \boldsymbol{x}_{i_0}^{N-j}$ yield the same N element combinations for all $0 \le j < N/2 - 1$; when

Algorithm 5: Codeword expansion function in PIRCOR*

1 **Function** `CodeExpandImp` $(\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\alpha-1}\})$
2 **for** $i \leftarrow 1$ **to** $\alpha - 1$ **do**
3 **for** $j \leftarrow 1$ **to** $\frac{N}{2}$ **do**
4 $\boldsymbol{x}_i^j = (\boldsymbol{x}_i[(0+j)\%N], \ldots, \boldsymbol{x}_i[(N-1+j)\%N])$;
5 Set $l = 0$;
  `// RSM-related expansion:`
6 **for** $i \leftarrow 0$ **to** $\alpha - 1$ **do**
7 **for** $j \leftarrow 1$ **to** $\frac{N}{2} - 1$ **do**
8 $\boldsymbol{y}_l = \boldsymbol{x}_i \circ \boldsymbol{x}_i^j$;
9 $l = l + 1$;
  `// RMM-related expansion:`
10 **for** *any* $0 \le i_0 < i_1 \le \alpha - 1$ **do**
11 **for** $j \leftarrow 0$ **to** $\frac{N}{2}$ **do**
12 $\boldsymbol{y}_l = \boldsymbol{x}_{i_0} \circ \boldsymbol{x}_{i_1}^j$;
13 $l = l + 1$;
14 **for** $j \leftarrow 1$ **to** $\frac{N}{2} - 1$ **do**
15 $\boldsymbol{y}_l = \boldsymbol{x}_{i_1} \circ \boldsymbol{x}_{i_0}^j$;
16 $l = l + 1$;
17 **return** β expanded codes $\{\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{\beta-1}\}$

$i_0 = i_1 = i$, only half of the N element combinations are distinct in $\boldsymbol{x}_i \circ \boldsymbol{x}_i^{N/2}$, the maximum rotation offset is then limited to $N/2$ to avoid duplication. Particularly,

- When $i_0 = i_1 = i$ (i.e., for the RSM), we use $N/2 - 1$ rotations of $\boldsymbol{x}_i$ ($0 < j < N/2$ in Line 7);
- When $i_0 < i_1$ (i.e., for the RMM), we use $\boldsymbol{x}_{i_1}$ and its $N/2$ rotations ($0 \le j \le N/2$ in Line 11) and $N/2 - 1$ rotations of $\boldsymbol{x}_{i_0}$ ($0 < j < N/2$ in Line 14).

- Although compared with PIRCOR, PIRCOR* decreases almost half rotation operations, in Lines 10-16 RMM generates $C_\alpha^2 \cdot N$ expanded codes, which is the same as that of PIRCOR for $\delta = 2$ (i.e., $C_\alpha^2 \cdot N$).

Totally, there are

$$\beta = \alpha \cdot \left(\frac{N}{2} - 1\right) + C_\alpha^2 \cdot N$$

expanded codewords in PIRCOR*. Thus, PIRCOR* produces more expanded codewords than PIRCOR, achieving a smaller query size and decreasing almost half rotation operations.

4.1.2 Primary codeword generation. Similarly, the primary codeword generation is revised in Algorithm 6.

Remark 4 (The reason for designing two alternative coding functions in Algorithm 6)**.** Recall that among all the β expanded codewords, only one has a Hamming weight of 1, while all the others have a Hamming weight of 0. Clearly, in order to guarantee security, the element 1 must appear in any codeword. Unlike PIRCOR, PIRCOR* expand codewords in two ways: RSM and RMM. Therefore, RSM and RMM both might generate such codeword with Hamming weight of 1, depending on the target index, as did in Lines 4-5 and 10-15.

Algorithm 6: Primary codeword generation in PIRCOR*

1 **Function** `RSMCode` (r, c)
2 **for** $l \in [\alpha]$ **do**
3 Set $\boldsymbol{x}_l = \boldsymbol{0}^N$;
4 Find unique $\phi \in [\alpha]$ and $\varphi \in \left[\frac{N}{2} - 1\right]$ such that $c = \phi \cdot (\frac{N}{2} - 1) + \varphi$;
5 Set $\boldsymbol{x}_\phi[r] = 1$ and $\boldsymbol{x}_\phi[(r + \varphi + 1)\%N] = 1$;
6 **return** $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\alpha-1}\}$;
7 **Function** `RMMCode` (r, c)
8 **for** $l \in [\alpha]$ **do**
9 Set $\boldsymbol{x}_l = \boldsymbol{0}^N$;
10 Find unique $\phi \in \left[C_\alpha^2\right]$ and $\varphi \in [N-1]$ such that
$$c = \phi \cdot N + \varphi$$
11 Retrieve the indices of two non-zero codewords in the ϕ-th indexed codeword combination in ascending order, i.e., $\phi = (\phi_0, \phi_1)$ where $\phi_0 < \phi_1 \le \alpha - 1$;
12 **if** $\varphi \in [0, \frac{N}{2}]$ **then**
13 Set $\boldsymbol{x}_{\phi_0}[r] = 1$ and $\boldsymbol{x}_{\phi_1}[(r + \varphi)\%N] = 1$;
14 **else if** $\varphi \in [\frac{N}{2} + 1, N)$ **then**
15 Set $\boldsymbol{x}_{\phi_0}[(r + \varphi - \frac{N}{2})\%N] = 1$ and $\boldsymbol{x}_{\phi_1}[r] = 1$;
16 **return** $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\alpha-1}\}$;

4.1.3 **Query**$(\mathrm{pp}, \beta, \mathrm{idx}) \to (\mathrm{query}, \mathbf{evk}, \mathbf{sk})$. The only difference in the query generation between PIRCOR and PIRCOR* lies in PIRCOR*'s additional mechanism that selects either `RSMCode` $(\cdot)$ or `RMMCode` $(\cdot)$ based on the column index c (Lines 4-7).

Algorithm 7: Query encoding in PIRCOR*

Input: The query index idx, the public parameter pp and the number of database column vectors β
Output: α query SIMD ciphertexts $\{\widetilde{\boldsymbol{x}}_0, \widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_{\alpha-1}\}$
1 Based on N and β, determine the query size α using Eq. (1) ;
2 $(\mathbf{sk}, \mathbf{evk}) \leftarrow \mathbf{KeyGen}(\mathcal{X})$;
3 Compute the row index $r := \mathrm{idx}\%N$;
4 Compute the column index $c := \lfloor \mathrm{idx}/N \rfloor$;
5 **if** $c \in [\alpha \cdot (N/2 - 1)]$ **then**
6 $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\alpha-1}\} \leftarrow$ `RSMCode` (r, c) ;
7 **else if** $c \in [\alpha \cdot (N/2 - 1), \beta)$ **then**
8 $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\alpha-1}\} \leftarrow$ `RMMCode` $(r, c - \alpha \cdot (N/2 - 1))$;
9 **for** $l \leftarrow 0$ **to** $\alpha - 1$ **do**
10 $\widetilde{\boldsymbol{x}}_l \leftarrow \mathbf{SIMDEnc}(\boldsymbol{x}_l)$
11 **return** $\{\widetilde{\boldsymbol{x}}_0, \widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_{\alpha-1}\}$;

4.1.4 **Answer**(query, **evk**, D) → resp. This algorithm differs from Algorithm 4 only in that the codeword expansion function (Line 1) is substituted with `CodeExpandImp(·)` that defined in Algorithm 5; all other procedures remain the same.

5 Implementation and Evaluation

5.1 Extensions and Optimizations

Our implementations apply the following extensions and optimizations for better concrete performances.

5.1.1 Pre-processing. The primary objective of preprocessing in PIRCOR and PIRCOR* is to shift computationally intensive operations to an offline stage, thereby minimizing the server's online response time. Unlike prior schemes like PIRANA [23], our additional computational overhead primarily stems from homomorphic rotations. Crucially, we observe that rotations can be entirely preprocessed offline without impacting online efficiency. This is achieved by leveraging the structural properties of LWE/RLWE ciphertexts.

A ciphertext in RLWE-based schemes (e.g., BFV [4]) comprises two components: ct_1 (uniformly random) and ct_2 (computed during encryption). By employing a common reference string (CRS) model, both the client and server can agree on ct_1 in advance. Consequently, the server precomputes operations involving ct_1 offline, significantly reducing online latency. On the other hand, the client only needs to transmit ct_2 part, halving the query size. Below, we detail how preprocessing applies to key operations in PIRCOR*:

- **PMult:** For $\widetilde{z} \leftarrow \mathbf{SIMDPMul}(\widetilde{x}, y)$, where $\widetilde{x} = (\mathrm{ct}_1, \mathrm{ct}_2)$, the result is $\widetilde{z} = (\boldsymbol{y} \cdot \mathrm{ct}_1, \boldsymbol{y} \cdot \mathrm{ct}_2)$. Since ct_1 is fixed via CRS, the server precomputes $\boldsymbol{y} \cdot \mathrm{ct}_1$ offline. Only $\boldsymbol{y} \cdot \mathrm{ct}_2$ is computed online, reducing PMult cost by 50%.
- **Rotation:** A rotation $\widetilde{z} \leftarrow \mathsf{SIMDRotate}(\mathrm{evk}, \widetilde{x}, j)$ involves two steps:

  $$\widetilde{z} = \psi_j(\mathrm{ct}_1) \odot \mathbf{evk} + (0, \psi_j(\mathrm{ct}_2)),$$

  where $\odot$ denotes gadget product (the dominant cost) and $\psi_j(\cdot)$ is an almost cost-free ring isomorphism. By precomputing $\psi_j(\mathrm{ct}_1) \odot \mathrm{evk}$ offline, the online phase only computes $\psi_j(\mathrm{ct}_2)$ and addition, eliminating almost all of the rotation overhead during response generation.
- **Mult:** For $\widetilde{z} \leftarrow \mathbf{SIMDMult}(\mathrm{evk}, \widetilde{x}, \widetilde{y})$, the operation expands as:

  $$\widetilde{z} = (\mathrm{ct}_1 \cdot \mathrm{ct}_1', \mathrm{ct}_1 \cdot \mathrm{ct}_2' + \mathrm{ct}_1' \cdot \mathrm{ct}_2) + (\mathrm{ct}_2 \cdot \mathrm{ct}_2') \odot \mathrm{evk}.$$

  where $\widetilde{x} = (\mathrm{ct}_1, \mathrm{ct}_2)$ and $\widetilde{y} = (\mathrm{ct}_1', \mathrm{ct}_2')$. The term $\mathrm{ct}_1\mathrm{ct}_1'$ (20% of total Mult cost in our parameter settings) is precomputed offline.

While the preprocessing method is applicable to all PIR protocols, it yields particularly significant gains for our PIRCOR and PIRCOR*. This stems from their unique structure where rotations - constituting the scheme's only additional overhead compared to other protocol such as PIRANA - are entirely ct_1-dependent and thus fully preprocessable.

5.1.2 Reordering ciphertext operations. As seen in Algorithm 4, the server must perform $\beta(\delta - 1)$ ciphertext-ciphertext multiplications, followed by β plaintext-ciphertext multiplications on the results. As is well known, the leveled FHE scheme - such as BFV in our implementation - employs a leveled computation architecture where higher-level operations incur greater latency. Since the computational cost of ciphertext-ciphertext multiplication is significantly higher than that of plaintext-ciphertext multiplication, we consider reordering these operations to improve computational efficiency. Specifically, we first perform plaintext-ciphertext multiplications under a high level (with large ciphertext modulus) and then carry out ciphertext-ciphertext multiplications on the results under the reduced level (Lines 6-10 in Algorithm 8).

5.1.3 Adaptive modulus switching. Compared to PIRANA [23], our PIRCOR and PIRCOR* introduce an additional $(\alpha - 1)\,(N - 1)$ and $\alpha \left\lfloor \frac{N}{2} - 1 \right\rfloor$ SIMD rotation operations, respectively. To reduce these computational costs, we first perform a modulus switching operation on $\widetilde{x}_i$ to lower the ciphertext modulus, followed by a SIMD rotation operation on the reduced-modulus ciphertext (Lines 1-4 in Algorithm 8).

Algorithm 8: Answer with computational optimization

Input: query = $(\widetilde{x}_0, \widetilde{x}_1, \ldots, \widetilde{x}_{\alpha-1})$, the evaluation key **evk**, and the database matrix D = $\{D_0, \ldots, D_{\beta-1}\}$

Output: The response $\widetilde{v}$, where the value of $r = [\mathrm{idx}\%N]$-th slot is $\mathrm{db}_{\mathrm{idx}}$ and the values of all other slots are 0s

1 **for** $i \leftarrow 1$ **to** $\alpha - 1$ **do**
2 $\widetilde{\omega}_i \leftarrow \mathbf{ModSwitch}(\widetilde{x}_i, q_1)$ ▷Reduce the size of $\widetilde{x}_i$;
3 **for** $j \leftarrow 1$ **to** $N - 1$ **do**
4 $\widetilde{\omega}_i^j \leftarrow \mathbf{SIMDRotate}(\mathbf{evk}, \widetilde{\omega}_i, j)$;
5 Set $l = 0$;
6 **for** *any* $0 \le i_0 < i_1 < \cdots < i_{\delta-1} \le \alpha - 1$ **do**
7 $\widetilde{y}_l \leftarrow \mathbf{SIMDPmul}\left(\widetilde{x}_{i_0}, D_l\right)$;
8 **for** *any* $0 \le j_1, j_2, \ldots, j_{\delta-1} \le N - 1$ **do**
9 $\widetilde{u}_l \leftarrow \mathbf{SIMDMul}(\mathbf{evk}, \widetilde{y}_l, \widetilde{\omega}_{i_1}^{j_1}, \ldots, \widetilde{\omega}_{i_{\delta-1}}^{j_{\delta-1}})$;
10 $l = l + 1$;
11 $\widetilde{v} = \mathbf{SIMDAdd}\left(\widetilde{u}_0, \widetilde{u}_1, \ldots, \widetilde{u}_{\beta-1}\right)$;
12 **return** $\widetilde{v}$;

5.1.4 Implementation optimization of SIMD rotation. In FHE SIMD encoding, the Galois group's automorphisms (e.g., $\psi_i : x \mapsto x^i$) rotate the slot contents. When the quotient group $\mathcal{H}$, defined as $\mathcal{H} = (\mathbb{Z}/2N\mathbb{Z})^*/G$ where G is the subgroup generated by the Frobenius automorphism ($x \mapsto x^p$), decomposes into a direct product of cyclic groups, rotations are occur along multiple independent dimensions. In our case, the rotation via automorphism has 2 dimensions, suppose an encoded message is structured as a 2D array:

$$\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \end{bmatrix} \tag{2}$$

Under a naive row-rotation via Galois automorphism (e.g., $\psi_3 : x \mapsto x^3$), the message becomes:

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_0 \\ x_5 & x_6 & x_7 & x_4 \end{bmatrix}$$

This does not achieve our code extension method desired cyclic left shift across all slots:

$$[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_0]$$

We propose two approach to solve this problem:

- *Computation-efficiency approach*: We construct the codewords of dimension N in two rows and determine the positions of the 1s based on the row index r. Specifically, we make the following revisions in PIRCOR, and similar modifications are applied to PIRCOR* accordingly.
  - In all Algorithms, set the maximum rotation offset to be $\frac{N}{2} - 1$.
  - Before Line 4 in `CodeGen(·)`, insert a line "$r^* = r/(N/2)$ and $r' = r\%(N/2)$".
  - In `CodeGen(·)`, Line 11 is revised as: $\boldsymbol{x}_l[r^* \frac{N}{2} + (r' + j_{k-1})\%\frac{N}{2}] = 1$.
- *Query-size-efficiency approach*: We combine the encoding method and the FHE column-rotation technique to revise `CodeExpand(·)` as follows.
  - For each codeword $\boldsymbol{x}_i$ of dimension N that is structured as Eq. (2), perform column-rotation (e.g., ψ_7) to swap rows to generate $\boldsymbol{x}_i^*$:

$$\begin{bmatrix} x_4 & x_5 & x_6 & x_7 \\ x_0 & x_1 & x_2 & x_3 \end{bmatrix}$$

    Then, for both $\boldsymbol{x}_i$ and $\boldsymbol{x}_i^*$, perform $N/2 - 1$ rotation operations.
  - For δ different codewords $\{\boldsymbol{x}_{i_0}, \ldots, \boldsymbol{x}_{i_{\delta-1}}\}$, where $0 \le i_0 < \cdots < i_1 \le \alpha - 1$, perform

$$\boldsymbol{y}_l = \boldsymbol{x}_{i_0} \circ \boldsymbol{x}_{i_1}^{j_1}(\text{ or } \boldsymbol{x}_{i_1}^{*\,j_1}) \circ \cdots \circ \boldsymbol{x}_{i_{\delta-1}}^{j_{\delta-1}}(\text{ or } \boldsymbol{x}_{i_{\delta-1}}^{*\,j_{\delta-1}})$$

    for $0 \le j_1, \ldots, j_{\delta-1} \le N/2 - 1$.
  - Before Line 4 in `CodeGen(·)`, insert a line "$r^* = r/(N/2)$ and $r' = r\%(N/2)$".
  - In `CodeGen(·)`, Line 11 is revised as: $j_k^* = j_k/(N/2)$, $j_k' = j_k\%(N/2)$; $\boldsymbol{x}_l[((r^* + j_k^*)\%2) \cdot \frac{N}{2} + (r' + j_{k-1}')\%\frac{N}{2}] = 1$.

Remark 5 (Comparison of two approaches)**.** We discuss a summary of the pros and cons of the two approaches in computational cost and query size.

(1) *Computational-efficiency approach*:

- *Pros*: Require a total of up to $\alpha(N-2)$ rotation operations[4], thereby achieving highest computational efficiency.
- *Cons*: The query size increases by a factor of $2^{\frac{\delta-1}{\delta}}$ for the same β compared to the *query-size-efficiency approach*.

(2) *Query-size-efficiency approach*:

- *Pros*: α N-dimensional codewords can still expand $\beta = C_\alpha^\delta N^{\delta-1}$ codes, achieving the highest codeword expansion. That is, given β, the query size $\alpha := O(\sqrt[\delta]{\delta!\beta/N^{\delta-1}})$ is the smallest.
- *Cons*: Require $\alpha(N-2)$ row-rotation operations and α column-rotation operations, which is larger than the *computational-efficiency approach*.

Depending on the specific application, one can adopt any of the above approaches to strike a balance among computational efficiency and communication overhead.

For short, we denote PIRCOR and PIRCOR* with these implementation optimizations as PIRCOR-op and PIRCOR*-op in the rest of this paper.

[4] In computational-efficiency approach, the maximum rotation offset is reduced from $N-1$ to $\frac{N}{2} - 1$, and thus only $\beta = C_\alpha^\delta \left(\frac{N}{2}\right)^{\delta-1}$ codewords are expanded. Compared to the *Query-size-efficiency approach*, the query size increases into $\alpha \cdot 2^{\frac{\delta-1}{\delta}}$ for the same β. Thus, the rotation operations are $\alpha \cdot 2^{\frac{\delta-1}{\delta}}(\frac{N}{2} - 1) \approx \alpha N - 2\alpha$ for large δ.

Table 1: Comparison of the asymptotic communication size, and computational complexity of the SOTA hintless single-server schemes.

	PIRCOR	**PIRANA** [23]	**YPIR** [28]
S. Que	$O(\sqrt{\lvert\text{DB}\rvert})$	$O(\sqrt{N\lvert\text{DB}\rvert})$	$O(N + \frac{\lvert\text{DB}\rvert}{N})$
S. Res	$O(N)$	$O(N)$	$O(N)$
C. Res-on	$O(\log N\lvert\text{DB}\rvert)$	$O(\log N\lvert\text{DB}\rvert)$	$O(\lvert\text{DB}\rvert)$
C. Res-off	–	–	$O(\frac{N}{\log N}\lvert\text{DB}\rvert)$

[a] "**S. Que**" and "**S. Res**" denote the query size and the response size; "**C. Res-on**" and "**C. Res-off**" denote the online response and offline response computational complexity, respectively.
[b] $\lvert\text{DB}\rvert$ is the number of elements in the database.
[c] N is the (R)LWE secret dimension.

5.2 Theoretical analysis

In this section, we present both an asymptotic comparison with the-state-of-the-art hintless single-server PIR schemes and a comprehensive evaluation of our proposed schemes.

Asymptotic Comparison. We summarize the asymptotic theoretical efficiency of PIRCOR in Table 1, where we compare it with the-state-of-the-art hintless single-server schemes: PIRANA [23] (representing coding-related PIR) and YPIR [28] (representing preprocessing-related PIR).

By using innovative REBC encoding design, PIRCOR reduces query size to $O(\sqrt{\lvert\text{DB}\rvert})$ – a $\sqrt{N}$ factor improvement over PIRANA and avoids YPIR's $O(N + \frac{\lvert\text{DB}\rvert}{N})$ linear scaling.

From the perspective of computational complexity, although YPIR performs well during the online phase, its total computational complexity (including **C. Res-on** and **C. Res-off**) is significantly higher than that of both PIRCOR and PIRANA. Furthermore, the linear query size of YPIR hinders its scalability when applied to large-scale databases. Note that we also give the concrete comparison in Appendix A.

5.3 Experimental results

Similar to PIRANA [23], we implement our PIRCOR and PIRCOR* as well as their their implementation optimized variants PIRCOR-op and PIRCOR*-op in C++ based on the Microsoft SEAL homomorphic encryption library (version 4.1.2) [5].

Parameter setting. Followed by PIRANA [23], we adopt the Brakerski-Fan-Vercauteren (BFV) homomorphic encryption scheme with the

[5] https://github.com/Microsoft/SEAL

polynomial degree $N = 8192$. The security parameters are configured according to SEAL library's 128-bit security recommendations. All cryptographic operations including ciphertext-plaintext multiplications and ciphertext-ciphertext are accelerated through Number Theoretic Transform (NTT) optimizations. Notably, and without loss of generality, we follow PIRANA [23] by setting $\delta = 2$ in all experiments. Then, for a given number of database elements (i.e., |DB|), the parameter α in PIRCOR (and PIRCOR-op) and PIRCOR* (and PIRCOR*-op) is determined by satisfying the following conditions:

- For PIRCOR & PIRCOR-op: $C_\alpha^2 \cdot N = \frac{|\mathrm{DB}|}{N}$.
- For PIRCOR* & PIRCOR*-op: $C_\alpha^2 \cdot N + \alpha \cdot (N/2-1) = \frac{|\mathrm{DB}|}{N}$.

To evaluate system scalability, we configure four database configurations with growing sizes: 128MB, 1GB, 8GB. Each configuration supports dual payload modes: 1 B, 8KB. Experimental results evaluate six critical performance metrics: query generation time (**C.Que**), offline response generation time (**C.Res-off**), online response time (**C.Res-on**), query size (**S. Que**), and response size (**S. Res**).

5.3.1 Concrete comparisons of our schemes. Based on theoretical analysis in Table 3, we provide concrete experimental results to confirm efficiency of our four schemes (PIRCOR, PIRCOR*, PIRCOR-op and PIRCOR*-op) in Table 2. We list three different database configurations, which are 128 MB, 1 GB and 8 GB with 1 B (small element size) and 8 KB (large element size), respectively.
(1) Client-side query size and time: Compared to the PIRCOR (or PIRCOR-op), PIRCOR* (or PIRCOR*-op) further reduces the query size due to the introduction of RSM. As discussed in Remark 3, RSM introduces $\alpha(N/2-1)$ additional expanded codewords compared to PIRCOR. Consequently, for a given |DB|, the value of α (i.e., the number of query codewords) in PIRCOR* is smaller than that in PIRCOR. Note that this advantage becomes more pronounced when the number |DB| of database elements is relatively small (i.e., resulting in a smaller α). However, as α increases, the benefit gradually diminishes and becomes negligible. For instance, when $|\mathrm{DB}| \in \{2^{14}, 2^{17}, 2^{20}\}$, PIRCOR* (or PIRCOR*-op) achieves a 2× reduction in query size. In contrast, for a larger database size such as $|\mathrm{DB}| = 2^{30}$, both PIRCOR* (or PIRCOR*-op) and PIRCOR (or PIRCOR-op) result in the same query size of 2.6 MB. This is because as α increases, $\alpha(N/2-1)$ becomes negligible compared to the number of codewords generated by RMM, i.e., $C_\alpha^2 N$. Similarly, as α varies, the query generation time exhibits the same trend.

Across all database configurations, the optimized variants, PIRCOR-op and PIRCOR*-op consistently halve the query size over their counterparts, PIRCOR and PIRCOR*, respectively. This is attributed to the pre-processing optimization discussed in Section 5.1.1, where the client only needs to transmit the query-related component ct_2 of the ciphertext $\widetilde{x} = (\mathrm{ct}_1, \mathrm{ct}_2)$.
(2) Server-side response size and time: Compared to PIRCOR, the PIRCOR* reduces the response time for small element sizes (e.g., for 1 B element size in 128 MB (1 GB), the online response time is 287 s (2086 s) for PIRCOR and 256 s (1961 s) for PIRCOR*), primarily due to the almost halved number of rotation operations. However, this advantage diminishes as the element size increases (like 8 KB). As shown in Table 3, in such case, the plaintext-ciphertext multiplications become the dominant cost, which is primarily determined by the database size. Therefore, under these circumstances, both schemes exhibit similar response times.

Across all database configurations, the optimized variants, PIRCOR-op and PIRCOR*-op, consistently reduce the online response time by up to 57%. For example, PIRCOR-op (or PIRCOR*-op) achieves a 57% (or 56%) reduction in online response time compared to PIRCOR (or PIRCOR*) on a 128 MB (or 8 GB) database with $2^{27} \times 1$ B. The significant performance gain stems from the optimized variants leveraging the pre-processing technique in Section 5.1.1, allowing all rotation operations, half of the plaintext-ciphertext multiplications, and 20% of the ciphertext-ciphertext multiplications to be offloaded to an offline phase. Moreover, our reordering ciphertext operation in Section 5.1.2 and adaptive modulus switching strategy in Section 5.1.3 further optimize the performance of ciphertext operations and rotation procedures. Note that the only difference between PIRCOR and PIRCOR* lies in the number of rotation operations. Since all rotations are handled in the offline phase in the optimized variants, i.e., PIRCOR-op and PIRCOR*-op, they achieve identical online response times across all database configurations.

5.3.2 Comparisons with State-of-The-Art PIR Schemes. In order to verify the theoritical analysis in Table 1, we provide concrete comparisons with PIRANA [23] and YPIR [28] in Tab, 2.
(1) Client-side query size and time: Across all database configurations, our four schemes, i.e, PIRCOR, PIRCOR*, PIRCOR-op and PIRCOR*-op, significantly outperform than PIRANA in terms of the query size. Notably, the advantages of our designs become increasingly prominent as |DB| scales. For instance, the query size of PIRANA is 60× larger than that of PIRCOR*-op ($|\mathrm{DB}| = 2^{27}$ in 128 MB) and **28**× larger than that of PIRCOR ($|\mathrm{DB}| = 2^{30}$ in 1 GB).

Under the small element size (e.g., 1 B), YPIR generally achieves smaller query sizes than our schemes. For 1 GB database ($2^{30} \times 1$B), YPIR achieves a query size of 0.8 MB, outperforming PIRCOR*-op (1.3 MB) by 38%. However, when |DB| is relatively small (2^{27}), our PIRCOR*-op (0.4 MB) outperforms YPIR (0.6 MB) by 38%. For large element sizes, our four schemes achieve substantial reductions in query size, with PIRCOR*-op demonstrating improvements by orders of magnitude. Moreover, the advantage becomes increasingly prominent as the database size grows. For an 8GB database ($2^{20} \times$ 8KB), our PIRCOR*-op reduces the query size to just 0.07 MB, outperforming YPIR by **99**% (compared to YPIR's 7.5 MB). As shown in Table 1, the primary reason is that the query size in our schemes depends solely on the number of database elements |DB|, whereas YPIR's query size depends on both the number and the size of the elements.

Note that in YPIR, in addition to query generation, an additional CDKS packing key [7] must be generated for the LWE-to-RLWE conversion, which is time-consuming. Therefore, although YPIR achieves smaller query sizes for small element sizes, Tab. 2 shows that our four schemes still offer significant improvements in query generation time compared to both PIRANA and YPIR. For an 8GB database ($2^{20} \times$ 8KB), our PIRCOR* (and PIRCOR*-op) achieves a query generation time of **5 ms**, outperforming PIRANA (47 ms) and YPIR (60.8 s) by factors of **9.4**× and **12,160**×, respectively. This acceleration derives from our scheme's optimized ciphertext expansion strategy, which minimizes required query ciphertexts.

Table 2: Experimental result comparison across PIRCOR variants, PIRANA and YPIR under different database configurations.

DB Size	Method	G.Que	G.Res-off	G.Res-on	|Que|	|Res|
128MB ($2^{27} \times 1$B)	**PIRCOR**	20 ms	-	287 s	1 MB	0.1 MB
	PIRCOR*	15 ms	-	256 s	0.9 MB	0.1 MB
	PIRCOR-op	20 ms	165 s	122 s	0.6 MB	0.1 MB
	PIRCOR*-op	15 ms	134 s	122 s	0.4 MB	0.1 MB
	PIRANA	500 ms	-	266 s	26 MB	0.1 MB
	YPIR	696 ms	5.64 s	0.07 s	0.6 MB	0.01 MB
1GB ($2^{30} \times 1$B)	**PIRCOR**	44 ms	-	2086 s	2.6 MB	0.1 MB
	PIRCOR*	44 ms	-	1961 s	2.6 MB	0.1 MB
	PIRCOR-op	44 ms	1106 s	980 s	1.3 MB	0.1 MB
	PIRCOR*-op	44 ms	981 s	980 s	1.3 MB	0.1 MB
	PIRANA	1500 ms	-	2182 s	74 MB	0.1 MB
	YPIR	2263 ms	32 s	0.2 s	0.8 MB	0.01 MB
128MB ($2^{14} \times 8$KB)	**PIRCOR**	10 ms	-	2 s	0.28 MB	0.1 MB
	PIRCOR*	5 ms	-	2 s	0.14 MB	0.1 MB
	PIRCOR-op	10 ms	1.1 s	0.9 s	0.14 MB	0.1 MB
	PIRCOR*-op	5 ms	1.1 s	0.9 s	0.07 MB	0.1 MB
	PIRANA	8 ms	-	2 s	0.44 MB	0.1 MB
	YPIR	885 ms	5 s	0.1 s	0.6 MB	0.04 MB
1GB ($2^{17} \times 8$KB)	**PIRCOR**	10 ms	-	16 s	0.28 MB	0.1 MB
	PIRCOR*	5 ms	-	16 s	0.14 MB	0.1 MB
	PIRCOR-op	10 ms	9 s	7 s	0.14 MB	0.1 MB
	PIRCOR*-op	5 ms	9 s	7 s	0.07 MB	0.1 MB
	PIRANA	19 ms	-	10 s	1 MB	0.1 MB
	YPIR	7 s	15.8 s	0.3 s	1.33 MB	0.04 MB
8GB ($2^{20} \times 8$KB)	**PIRCOR**	10 ms	-	129 s	0.28 MB	0.1 MB
	PIRCOR*	5 ms	-	128 s	0.14 MB	0.1 MB
	PIRCOR-op	10 ms	72 s	57 s	0.14 MB	0.1 MB
	PIRCOR*-op	5 ms	71 s	57 s	0.07 MB	0.1 MB
	PIRANA	47 ms	-	73 s	3 MB	0.1 MB
	YPIR	60.8 s	100.8 s	1.4 s	7.5 MB	0.04 MB

[a] "**G.Que** ", "**G. Res-off** ", "**G. Res-on** ", "**|Que|**" and "**|Res|**" denote the query generation time, offline-response time, online-response time, the query size and the response size, respectively. "PIRCOR-op" and "PIRCOR*-op" represent the original schemes with the implementation optimizations in Section 5.1.

(2) Server-side response size and time: From Tab. 2, we can see our four schemes and PIRANA achieve the same response size, while larger than that of YPIR. Meanwhile, YPIR achieves the best response time, especially for online cost. Nevertheless, when the database is small but the element size is large (e.g., 128 MB, $2^{14} \times 8$ KB), the performance gap between YPIR and our PIRCOR*-op narrows considerably.

Compared to PIRANA, our PIRCOR and PIRCOR* exhibit longer response times, primarily due to the additional rotation operations required by our encoding strategy. Moreover, with the increase of |DB|, the relative response time overhead of our schemes compared to PIRANA progressively reduces — for instance in PORCOR, shrinking from 50% (2^{20}) to as low as 7% (2^{27}). Thanks to the implementation optimization techniques, the optimized schemes (PIRCOR-op and PIRCOR*-op) achieve the best response efficiency across all database configurations, reducing the response time up to 50% compared to PIRANA.

(3) Comparison conclusion: Overall, our schemes are more efficient in terms of query size and generation time, whereas YPIR performs better in response time with pre-processing, and PIRANA shows advantages in response time without pre-processing. Thus, in scenarios where pre-processing is supported, our PIRCOR*-op is ideal for settings like mobile cloud search, where clients have limited resources and servers are powerful, whereas YPIR performs better in the reverse scenario. Likewise, in the absence of pre-processing, PIRCOR (PIRCOR* for $\delta = 2$) is preferable for resource-limited clients.

(4) Theoretical-Practical Discrepancy Analysis: Regarding online response time and the response size, both PIRCOR*-op and PIRANA demonstrate inferior performance compared to YPIR due

to inherent constraints of BFV homomorphic encryption. While our scheme follow the standard SEAL library implementation, YPIR employs customized optimizations for its underlying LWE homomorphic encryption realization. Our scheme further requires sequential plaintext multiplication, rotation, and ciphertext multiplication operations, consuming 4 levels for RLWE ciphertext polynomials with coefficient modulus. This necessitates a polynomial degree $N = 8192$ versus YPIR's $N = 1024$, consequently increasing homomorphic computation costs and ciphertext size. Direct comparion of computational overhead and response sizes under these parameter configurations would inherently disadvantage our scheme. Crucially, however, theoretical analysis in Table 1 confirms that our scheme achieves response sizes equivalent to YPIR while the overall computational complexity is superior to YPIR.

In the subsequent work, we plan to implement undering optimizations matching YPIR' engineering efforts, thereby enabling performance comparisons under identical parameter configurations for a more equitable evaluation of our scheme's efficiency.

6 Conclusion

In this paper, we introduce two hintless single-server PIR schemes, PIRCOR and PIRCOR*, designed to reduce both the query size and the corresponding query generation time. PIRCOR achieves a query size reduction by a factor of $O\left(N^{\frac{\delta-1}{\delta}}\right)$ compared to PIRANA [23], at the expense of an additional $\alpha(N-1)$ ciphertext rotations. Building on PIRCOR, PIRCOR* further reduces the number of rotations by 50%, and even achieves a smaller query size when $\delta = 2$. To improve online response time, we also present optimized variants — PIRCOR-op and PIRCOR*-op — which shift query-independent computation into an offline phase. The online response time in both optimized variants is 2× faster than PIRANA.

References

[1] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. 2021. Addra: Metadata-private voice communication over fully untrusted infrastructure. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16*.

[2] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. 2021. Communication-Computation Trade-offs in PIR. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13*. 1811–1828.

[3] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy, SP 2018, 21-23 May, San Francisco, California, USA*. 962–979.

[4] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23*, Vol. 7417. 868–886.

[5] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25*. 97–106.

[6] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18*, Vol. 6841. 505–524.

[7] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2021. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In *Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part I*, Vol. 12726. 460–479.

[8] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, Part I*, Vol. 10820. 360–384.

[9] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7*, Vol. 10624. 409–437.

[10] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. 1995. Private Information Retrieval. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October*. 41–50.

[11] Henry Corrigan-Gibbs and Dmitry Kogan. 2020. Private Information Retrieval with Sublinear Online Time. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, Part I*, Vol. 12105. 44–75.

[12] Alex Davidson, Gonçalo Pestana, and Sofía Celi. 2023. FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval. *Proc. Priv. Enhancing Technol.* 2023, 1 (2023), 365–383.

[13] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.* (2012), 144.

[14] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2*. 169–178.

[15] Ashrujit Ghoshal, Mingxun Zhou, and Elaine Shi. 2024. Efficient Pre-processing PIR Without Public-Key Cryptography. In *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, Vol. 14656. 210–240.

[16] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11*. 3889–3905.

[17] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. 2019. Mobile Private Contact Discovery at Scale. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16*. 1447–1464.

[18] Dmitry Kogan and Henry Corrigan-Gibbs. 2021. Private Blocklist Lookups with Checklist. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. 875–892.

[19] Eyal Kushilevitz and Rafail Ostrovsky. 1997. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22*. 364–373.

[20] Arthur Lazzaretti and Charalampos Papamanthou. 2023. TreePIR: Sublinear-Time and Polylog-Bandwidth Private Information Retrieval from DDH. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, Part II*, Vol. 14082. 284–314.

[21] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz. 2024. Hintless Single-Server Private Information Retrieval. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, Part IX*, Vol. 14928. 183–217.

[22] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. 2019. Protocols for Checking Compromised Credentials. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15*. 1387–1403.

[23] Jian Liu, Jingyu Li, Di Wu, and Kui Ren. 2024. PIRANA: Faster Multi-query PIR via Constant-weight Codes. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23*. 4315–4330.

[24] Ming Luo, Feng-Hao Liu, and Han Wang. 2024. Faster FHE-Based Single-Server Private Information Retrieval. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18*. 1405–1419.

[25] Rasoul Akhavan Mahdavi and Florian Kerschbaum. 2022. Constant-weight PIR: Single-round Keyword PIR via Constant-weight Equality Operators. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12*. 1723–1740.

[26] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR : Private Information Retrieval for Everyone. *Proc. Priv. Enhancing Technol.* 2016, 2 (2016), 155–174.

[27] Samir Jordan Menon and David J. Wu. 2022. SPIRAL: Fast, High-Rate Single-Server PIR via FHE Composition. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26*. 930–947.

[28] Samir Jordan Menon and David J. Wu. 2024. YPIR: High-Throughput Single-Server PIR with Silent Preprocessing. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16*.

[29] Prateek Mittal, Femi G. Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. 2011. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *20th USENIX Security Symposium, San Francisco, CA,*

USA, August 8-12.

[30] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response Efficient Single-Server PIR. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19*. 2292–2306.

[31] Ling Ren, Muhammad Haris Mughees, and I Sun. 2024. Simple and Practical Amortized Sublinear Private Information Retrieval using Dummy Subsets. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18*. 1420–1433.

[32] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. 2019. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16*. 1556–1571.

[33] Ke Coby Wang and Michael K. Reiter. 2020. Detecting Stuffing of a User's Credentials at Her Own Accounts. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14*. 2201–2218.

[34] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. 2024. Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23*. 4296–4314.

[35] Jinbao Zhu, Qifa Yan, Xiaohu Tang, and Ying Miao. 2021. Capacity-Achieving Private Information Retrieval Schemes From Uncoded Storage Constrained Servers With Low Sub-Packetization. *IEEE Trans. Inf. Theory* 67, 8 (2021), 5370–5386.

A Appendix for Algorithm 9

Algorithm 9: Answer (large size element $|\mathrm{db}_i| = \theta \cdot N \cdot |p|$)

Input: query = $(\widetilde{\boldsymbol{x}}_0, \widetilde{\boldsymbol{x}}_1, \ldots, \widetilde{\boldsymbol{x}}_{\alpha-1})$, the evaluation key $\mathbf{evk}$, and the database column vectors D^*

Output: The response resp = $(\widetilde{\boldsymbol{v}}_0, \widetilde{\boldsymbol{v}}_1, \ldots, \widetilde{\boldsymbol{v}}_{\theta-1})$

Run Line 1 in Algorithm 4 to obtain $(\widetilde{\boldsymbol{y}}_0, \widetilde{\boldsymbol{y}}_1, \ldots, \widetilde{\boldsymbol{y}}_{\beta-1})$;
Initialize $(\widetilde{\boldsymbol{u}}_0, \ldots, \widetilde{\boldsymbol{u}}_{\theta N-1})$: each $\widetilde{\boldsymbol{u}}_i$ is an encryption of $\mathbf{0}^N$;
for $l \leftarrow 0$ **to** $\beta - 1$ **do**
    for $i \leftarrow 0$ **to** $\theta N - 1$ **do**
        $\widetilde{f_i} \leftarrow \mathbf{SIMDPmul}\left(\widetilde{\boldsymbol{y}}_l, D_{l,i}\right)$;
        $\widetilde{\boldsymbol{u}}_i \leftarrow \mathbf{SIMDAdd}(\widetilde{f_i}, \widetilde{\boldsymbol{u}}_i)$;
Initialize $(\widetilde{\boldsymbol{v}}_0, \widetilde{\boldsymbol{v}}_1, \ldots, \widetilde{\boldsymbol{v}}_\theta)$: each $\widetilde{\boldsymbol{v}}_i$ is an encryption of $\mathbf{0}^N$;
for $i \leftarrow 0$ **to** $\theta - 1$ **do**
    for $j \leftarrow 0$ **to** $N - 1$ **do**
        $\widetilde{\boldsymbol{v}}_i \leftarrow \mathbf{SIMDAdd}(\widetilde{\boldsymbol{v}}_i, \widetilde{\boldsymbol{u}}_{iN+j})$;
        $\widetilde{\boldsymbol{v}}_i \leftarrow \mathbf{SIMDRotate}(\mathbf{evk}, \widetilde{\boldsymbol{v}}_i, 1)$;
return resp = $(\widetilde{\boldsymbol{v}}_0, \widetilde{\boldsymbol{v}}_1, \ldots, \widetilde{\boldsymbol{v}}_{\theta-1})$;

B Concrete Comparison of Theoretical analysis

Furthermore, the shared ciphertext structure and homomorphic operations between PIRCOR and PIRANA (but different with YPIR) enable a unified framework for fine-grained (not just asymptotic) comparisons. It allows us to compare PIRCOR, PIRCOR*, and their optimized variants with PIRANA, isolating the efficiency gains introduced by our novel encoding techniques and implementation optimizations, as summarized in Table 3.

The analysis of Table 3 reveals significant improvements in communication and computational efficiency achieved by PIRCOR and its variants. In terms of query size, PIRCOR reduces it by a factor of $\sqrt{N}$ compared to PIRANA, owing to our novel Rotation-based Expanded Binary Code (REBC). The optimized variants (PIRCOR-op and PIRCOR$*$-op) further halve the query size by offloading ciphertext components (e.g., $\mathbf{ct}_1$) to a preprocessing phase. While REBC introduces additional homomorphic rotation operations, the unique structure of homomorphic automorphism enables nearly *full offline computation* via preprocessing. Finally, PIRCOR and its variants maintain identical response sizes, ensuring minimal communication cost. This combination of REBC-driven compression and implementation optimization demonstrates a practical PIR solution: achieving sublinear query growth ($O(\sqrt{|\mathrm{DB}|})$) while matching or exceeding the performance of state-of-the-art protocols.

Table 3: Concrete Comparison of our schemes and PIRANA [23].

		PIRCOR	**PIRCOR***	**PIRCOR-op**	**PIRCOR*-op**	**PIRANA**
S. Que		$\sqrt{2\|\mathrm{DB}\|}$	$\sqrt{2\|\mathrm{DB}\|}$	$\sqrt{\|\mathrm{DB}\|/2}$	$\sqrt{\|\mathrm{DB}\|/2}$	$\sqrt{2\|\mathrm{DB}\|N}$
S. Res		$2N$	$2N$	$2N$	$2N$	$2N$
C. Res-on	# Rotate	$\sqrt{2\|\mathrm{DB}\|}$	$\sqrt{\|\mathrm{DB}\|/2}$	–	–	–
	# Pmul	$\|\mathrm{DB}\|/N$	$\|\mathrm{DB}\|/N$	$\|\mathrm{DB}\|/(2N)$	$\|\mathrm{DB}\|/(2N)$	$\|\mathrm{DB}\|/N$
	# Mul	$\|\mathrm{DB}\|/N$	$\|\mathrm{DB}\|/N$	$0.8\|\mathrm{DB}\|/N$	$0.8\|\mathrm{DB}\|/N$	$\|\mathrm{DB}\|/N$
C. Res-off	# Rotate	–	–	$\sqrt{2\|\mathrm{DB}\|}$	$\sqrt{\frac{\|\mathrm{DB}\|}{2}}$	–
	# Pmul			$\|\mathrm{DB}\|/(2N)$	$\|\mathrm{DB}\|/(2N)$	
	# Mul			$0.2\|\mathrm{DB}\|/N$	$0.2\|\mathrm{DB}\|/N$	

[a] $|\mathrm{db}| = |p|$ represent the element size; "Rotate", "Pmul" and "Mul" are the ciphertext rotation, plaintext-ciphertext multiplication and ciphertext-ciphertext multiplication, respectively.
[b] "**PIRCOR-op**" and "**PIRCOR*-op**" are **PIRCOR** and **PIRCOR*** with the implementation optimizations in Section 5.1, respectively.