Scalable Multi-Server Private Information Retrieval using Homomorphic Encryption for Cloud Environments

Akshit Aggarwal[a,*], Pulkit Bharti[a], Yang Li[b] and Srinibas Swain[c]

[a]Department of Computer Science and Engineering, IIIT Guwahati, Guwahati, 781015, Assam, India
[b]Deakin University, 221 Burwood Highway, Burwood, Victoria 3125, Melbourne, Australia
[c]Adelaide University, School of Computer Science and Information Technology, Adelaide, 5000, South Australia, Australia

ARTICLE INFO

Keywords:
Cloud Computing
Fully Homomorphic Encryption (FHE)
Multi-Server
Private Information Retrieval (PIR)

ABSTRACT

Private Information Retrieval (PIR) based on Fully Homomorphic Encryption (FHE) enables secure query processing over encrypted data without revealing user intent. However, existing FHE based PIR solutions suffer from significant computational overhead and often shift storage and post-processing burdens to the client, limiting their applicability in large scale cloud environments. In this paper, we propose a scalable multi-server PIR framework designed for distributed cloud systems, which efficiently redistributes computation, storage, and data organization across three non-colluding servers. The first server performs homomorphic query evaluation over compact metadata and data location indices, reducing the need for direct interaction with large datasets. The second server constructs an integrated and randomized result representation by combining homomorphically evaluated ciphertexts with symmetrically encrypted location information, thereby preventing access pattern leakage while eliminating client-side storage overhead. The third server employs a Restricted Bin Packing Algorithm (RBA) to organize data efficiently across clustered storage, enabling balanced data distribution and improved retrieval performance. The proposed architecture minimizes redundant homomorphic operations, reduces communication overhead, and significantly improves system scalability. We implement the framework using the TenSEAL library and evaluate it on a real-world dataset. Experimental results demonstrate an average 4.38× reduction in query execution time compared to a representative FHE based PIR baseline, along with substantial reductions in client-side storage and communication costs. These results highlight the effectiveness of the proposed approach in enabling efficient, scalable, and privacy-preserving query processing suitable for modern distributed cloud and high-performance computing environments.

1. Introduction

Cloud based platforms host massive volumes of publicly accessible information, enabling users to search and retrieve data at an unprecedented scale. While such platforms significantly improve accessibility, they also raise serious privacy concerns when users issue queries related to sensitive topics. In many practical scenarios, the data itself is public, yet the act of querying and retrieving specific information can reveal sensitive user intent. Preserving query privacy and access pattern confidentiality in large scale cloud environments therefore remains a critical challenge [1, 2].

Private Information Retrieval (PIR) allows a client to retrieve data items from a server without revealing which item is being accessed [3]. PIR is widely used in many areas, such as private location delivery [4], anonymous communication [5, 6], private voting [7], and privacy preserving recommendation services [8, 9]. However, existing PIR solutions face significant limitations when deployed in large scale distributed cloud environments. Single server computational PIR schemes rely on expensive cryptographic operations, leading to substantial server side computation and monetary overhead [10, 11, 12]. To reduce this cost, many approaches implicitly shift post processing and storage responsibilities to the client, requiring users to download and locally handle large volumes of encrypted data. Such designs are impractical for real world cloud deployments where clients are resource constrained and scalability is essential.

Multi server PIR architectures provide a promising alternative by distributing computation and storage responsibilities across multiple non colluding servers [5, 13, 14]. By separating system functionality across servers, multi server approaches can reduce computational bottlenecks and improve scalability in clustered environments. However, designing such systems introduces new challenges, including preventing access pattern leakage, minimizing redundant homomorphic operations, and avoiding excessive client side storage while maintaining strong query privacy guarantees.

In this work, we propose a distributed multi server framework for privacy preserving query retrieval that is specifically tailored for cloud and clustered environments. The framework separates query evaluation, result storage, and data organization across three non colluding servers. The first server stores query related metadata and data location indices and performs homomorphic query evaluation over the metadata. The evaluated query outcomes, together with the corresponding data location information encrypted using a symmetric key scheme, are forwarded to the second server. The second server stores an integrated result representation that combines homomorphically evaluated ciphertexts and symmetrically encrypted data location information in a randomized layout. This design prevents access pattern leakage

*Corresponding author
akshitaggarwal20jan@gmail.com (A. Aggarwal); pulkit.bharti21b@iiitg.ac.in (P. Bharti); kelvin.li@deakin.edu.au (Y. Li); srinibas.swain@adelaide.edu.au (S. Swain)
ORCID(s): 0000-0003-0348-7304 (A. Aggarwal)

while eliminating the need for client side result storage. The third server organizes the dataset using a Restricted Bin Packing Algorithm, which enables balanced storage and efficient retrieval in clustered storage systems. This architectural separation reduces server side homomorphic computation, minimizes client overhead, and improves overall scalability. We implement the proposed framework using TenSEAL library [15] and evaluate it on a real world dataset. Experimental results show that the proposed approach achieves a 4.38× reduction in query execution time compared to a representative FHE based PIR baseline [13], while requiring fewer homomorphic operations and significantly reducing client side storage. These results demonstrate that the proposed framework is practical, scalable, and well suited for deployment in clustered cloud environments.

1.1. Related Work

In this section, we discuss the relevant literature in the area of private information retrieval.

To the best of our knowledge, PIR was first introduced by Chor et al. [3] by considering a multi-server based approach where replicas of data were stored on each server. Later on, Kushilevitz et al. [16] introduced the first single server based PIR protocol. The main limitation of previous works was high communication complexity [3, 16, 17]. To address the aforementioned limitation, Yi et al. [18] proposed a Fully Homomorphic Encryption (FHE) based PIR scheme, achieving logarithmic communication complexity with respect to database size. However, the main limitation of their work is high computational overhead on the server. Reducing server side computation overhead introduces several drawbacks [10, 19, 20, 13]. A few of them are discussed as follows.

First, a few authors have proposed batch based encoding scheme (as batch encoding helps to apply multiple operations simultaneously, which reduces the server computation overhead) where the authors send multiple queries in a single slot [10, 19, 11]. The main limitation of their work is lack of correctness guarantees for query results (as batching includes packing and unpacking of ciphertexts, which may lose some of the functionality of either query or database) [19, 11]. To improve it, a few authors have proposed a multi-server based approach to achieve correct query results [13, 21, 14]. The work by Ahmad et al. [13] proposed three servers (ideally interaction rounds between client and server in [13]) protocol where each server is responsible for distinct tasks and retrieves the relevant document. The main drawback of [13]'s scheme is oblivious RAM (where receiver gets one of many options secretly) that increases the client side computation. Aggarwal et al. [21] proposed a two server approach in which the first server stores metadata while the second server stores the dataset. The primary limitation of this approach is query leakage, along with inadequate client-side storage management. Furthermore, Gibbs et al. [14] proposed an adaptive query (which depends on the response of the previous query) based approach where server is divided into a few chunks (ideally multi-servers) with query leakage problems. Later, a few authors proposed a single server based approach to overcome the multi-server issues. The work by [12, 20, 22] performs various steps with the help of single server. The work in [12] used Fermat's Little Theorem to convert each vector into either an encryption of one or zero, which is helpful for retrieving the record corresponding to the encryption of one. In contrast, [20] used asymmetric scalar-product-preserving encryption (ASPE) [23], which is helpful for retrieving the first k similar documents. Additionally, [22] used additive sharing of random vectors, providing authentication to the server. The main limitation of these works is high number of homomorphic operations (which is time consuming). Further, a few authors first download the database and then search for the desired query on the database, which increases the client storage [24, 25, 26]. While downloading the database, to turn the communication channel secure, the work by [25, 26] introduces learning with error (LWE) based assumption [27], which introduces noise in the database which is redundant. Thus, reducing server computation faces the following challenges: 1) inaccurate query results, 2) query leakage, 3) a large number of homomorphic operations, and 4) increased client storage requirements. These limitations motivate the need for a PIR design that simultaneously reduces server computation overhead, prevents access pattern leakage, and avoids excessive client side storage requirements. Table 1 summarizes a few notable works.

The comparison in Table 1 is qualitative in nature, as exact quantitative metrics such as the number of homomorphic operations and client-side storage depend on system parameters and implementation details across different works.

1.2. Our Contribution

This paper introduces a scalable and privacy preserving PIR framework designed for deployment in clustered cloud environments. The main contributions of this work are summarized as follows.

- We propose a three server PIR architecture that cleanly separates query evaluation, result storage, and data organization. Unlike prior FHE based PIR schemes that overload either the server or the client, the proposed design reduces server side homomorphic computation while eliminating the need for client side result storage.
- We introduce a hybrid cryptographic integration across servers, where homomorphic encryption is used for privacy preserving query evaluation, and symmetric key encryption is employed for protecting data location information. The evaluated results are stored in a randomized layout at an intermediate server, effectively preventing access pattern leakage without relying on oblivious RAM or expensive client interaction.
- We design and apply a Restricted Bin Packing Algorithm (RBA) for organizing data on storage server. RBA enables balanced data placement and efficient

Work(s)	Architecture	Query Leakage	Interaction Rounds	FHE Computation Overhead	Client-Side Storage Requirement
[12]	Single-server	×	2	High	Low
[13]	Multi-server	×	6	High	High
[14]	Single-server	✓	2	Low	Low
[20]	Single-server	✓	3	Low	Low
[21]	Multi-server	✓	4	Low	High
[25]	Single-server	×	2	Low	High
Our Work	Multi-server	×	5	Reduced	Low

Table 1
Qualitative comparison of existing PIR schemes based on architecture, interaction rounds, computational overhead, and client-side storage.

retrieval in clustered storage systems, reducing storage fragmentation and improving scalability under large datasets.

- We implement the proposed framework using TenSEAL library [15] and conduct extensive experimental evaluation. Results demonstrate a 4.38× improvement in query execution time over a representative FHE based PIR baseline, while significantly reducing homomorphic operations, client side storage.

1.3. Organization of our paper

In Section 2, we present preliminaries, followed by proposed methodology in Section 3. Section 4 discussed the implementation details and performance evaluation of our work. Later, we discuss the security of our work in Section 5. We discuss the limitations of our work in Section 6. Finally, the conclusion of our work is shown in Section 7.

2. Preliminary

In this section, we first discuss BFV homomorphic encryption which is used for encryption and decryption of client's query. Thereafter, we discuss the bin packing algorithm which is used for storage of data.

2.1. BFV Homomorphic Encryption

BFV encryption enables simultaneous operations on a data vector and leverages the Single Instruction Multiple Data (SIMD) programming model. Each plaintext element from $\mathbb{Z}_p$ (p is a prime number) gains noise when encrypted, and all operations converting plaintext to ciphertext preserve value modulo p.

- **Add**(ct_1,ct_2) consists of two ciphertexts that contain the encryption of plaintext m_1, m_2, and returns an encryption of $(m_1 + m_2)$ (component-wise addition).
- **AddPlain**(ct_1,m_2) consists of ciphertext (encryption of m_1), plaintext m_2, and returns an encryption of $(m_1 + m_2)$ (component-wise addition).
- **Subtract**(ct_1,ct_2) consists of two ciphertexts that contain the encryption of plaintext m_1, m_2, and returns an encryption of $(m_1 - m_2)$ (component-wise subtraction).
- **SubtractPlain**(ct_1,m_2) consists of ciphertext (encryption of m_1), plaintext m_2, and returns an encryption of $(m_1 - m_2)$ (component-wise subtraction).
- **Mult**(ct_1,ct_2) consists of two ciphertexts that contain the encryption of plaintext m_1, m_2, and returns an encryption of $(m_1 \times m_2)$ (component-wise multiplication).
- **MultPlain**(ct_1,m_2) consists of ciphertext (encryption of m_1), plaintext m_2, and returns an encryption of $(m_1 \times m_2)$ (component-wise multiplication).

2.2. Bin Packing Algorithm

Bin packing is a combinatorial optimization problem where the objective is to pack a collection of items into a minimum number of bins, such that each bin has a fixed capacity. The key constraint is that the sum of the weights of the items in any bin must not exceed the bin's capacity. This problem is *NP-hard*, and various heuristic and exact algorithms have been proposed to tackle it (refer to [28] for a detailed discussion on bin packing). In this study, we extend the basic bin packing algorithm by incorporating additional constraints, as discussed in Algorithm 3. These constraints further refine the packing process, ensuring that solution satisfies specific additional requirements not considered in standard formulation.

3. Proposed Methodology

This section describes the proposed approach of this paper. We propose an FHE based private information retrieval protocol in which the client encrypts its query and sends it to server. Server processes the encrypted query and returns the relevant data without learning the query content. To achieve secure and efficient data retrieval, we adopt a three-server architecture, where all servers are assumed to be non-colluding and semi-honest (a formal threat model and security goals

are presented in Section 5.1).[1] The architecture of servers are discussed as follows.

1. **Server 1** stores metadata describing the dataset, including which categories are present and the location identifiers of the corresponding data on Server 3. This enables Server 1 to homomorphically match the encrypted query to its relevant category.
2. **Server 2** stores computation results obtained from Server 1. The data in Server 2 is stored in a random order (that is, the first entry from Server 1 does not necessarily appear in the first entry of Server 2) which prevents access pattern leakage.
3. **Server 3** acts as main memory, storing actual dataset. To optimize storage, dataset is arranged according to Restricted Bin Packing Algorithm (RBA) (as discussed in Algorithm 3).

First, homomorphic computation is performed on Server 1 to match client's encrypted query with stored category data. Thereafter, the corresponding location identifier is encrypted using symmetric encryption scheme (say, Advanced Encryption Standard (AES) [29], though the details are beyond the scope of this paper) and sent to Server 2. Server 2 decrypts the homomorphic computation results; if decryption result is zero, indicating the presence of client's query and forwards the corresponding encrypted location to client. Finally, the client decrypts the location and retrieves the relevant web links from Server 3 without revealing any query or access information (as shown in Figure 1). The detailed analysis of our proposed approach are as follows.

3.1. Key Sharing Mechanism

The client first shares a secret key (K') with Server 1 over a secure channel, which is used exclusively for encrypting the link locations (say, AES encryption of $Loc[i, j]$) before sending them to Server 2. This ensures that Server 2 can forward encrypted locations to the client without learning the actual location values. The use of symmetric encryption is motivated by its computational efficiency compared to asymmetric cryptographic schemes. Since a large number of location identifiers are encrypted and transmitted, symmetric encryption provides lower latency and reduced computational overhead. Additionally, pre-sharing of keys avoids the need for public-key operations during query processing. Thereafter, the client shares another secret key (K) with Server 2 over a secure channel. Key K is used solely for homomorphic encryption and decryption of query vectors and computation results, enabling Servers 1 and 2 to process encrypted queries without learning their content.

3.2. Server 1: Existence of entered query

Server 1 is helpful for checking whether client query q is present in the server or not (as shown in Algorithm 1).

[1] Security relies on the standard non-collusion assumption, under which no single server learns both the plaintext query and the corresponding data locations.

Algorithm 1 Existence of Entered Query with Encrypted Location

Input: Query q, secret keys K and K', Server 1 with stored web links locations $Loc[i, j]$
Output: Homomorphic computed results r_i and encrypted location $\text{Enc}_{K'}(Loc[i, j])$
Procedure:
Client Side:
Expand q to form the query vector q' covering all categories
Encrypt the query vector: $\text{Enc}_K(q')$
Send $\text{Enc}_K(q')$ to Server 1
Server 1 Side:
for each web link W_l and corresponding location $Loc[i, j]$ stored on Server 1 **do**
    Compute $d_i = \text{Enc}_K(q') - W_l$ {Homomorphic subtraction}
    Compute $r_i = d_i \cdot d_i$ {Homomorphic multiplication}
    Encrypt the location: $\text{Enc}_{K'}(Loc[i, j])$
end for
Send $\{r_i\}$ and $\{\text{Enc}_{K'}(Loc[i, j])\}$ to Server 2

First, to perform homomorphic operations, client expands its query vector to include all categories (in our case, a total of five) (say q'). Later, client encrypts its query vector using BFV encryption (say $Enc_K(q')$) and sends it to server. Server homomorphically checks the encrypted query from each web link (where the client encrypted vector is subtracted from each link). After performing homomorphic subtraction, server homomorphically multiplies the resulting vectors and returns the final result along with the corresponding encrypted location (say $Enc_{K'}(Loc[i, j])$) to Server 2 (as shown in Figure 2). The following toy example illustrates the above observation.

Example 1 (Based on Server 1)

1. **Client Side:**
   (a) To facilitate element wise homomorphic operations with the stored link vectors, client expands the query into a vector q' of length equal to the number of categories. This ensures that subtraction and multiplication can be applied component wise without modifying homomorphic computation routines, even though all entries have the same value.

$$q' = [3, 3, 3, 3, 3]$$

   (b) The client encrypts the query vector:

$$Enc_K(q') = [Enc_K(3),\ Enc_K(3),\ Enc_K(3),\ Enc_K(3),\ Enc_K(3)]$$

   (c) The encrypted query vector is sent to Server 1.
2. **Server Side:**

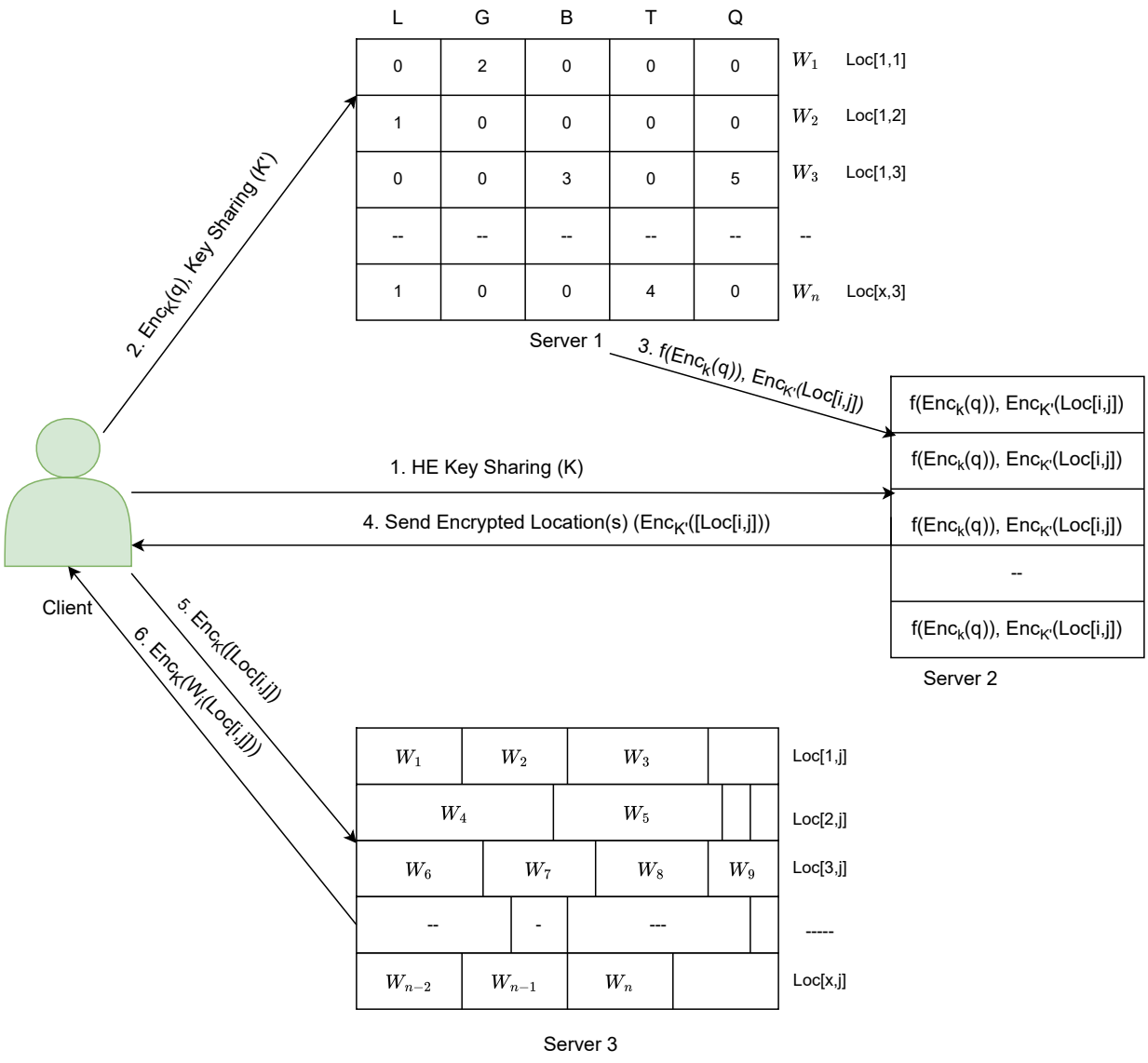

Figure 1: Here, K' and K denote the secret keys for secure and homomorphic encryption, respectively. Digits "1–5" indicate the presence of L, G, B, T, and Q categories in Server 1, while "0" indicates absence. W_k $(1 \leq k \leq n)$ represents web link k, and $Loc[i, j]$ denotes its position in the jth slot of bin i $(i, j > 0)$. The client query is q, and Server 2 stores the results computed by Server 1. $f(Enc_K(q))$ denotes computation over encrypted query, $Enc_{K'}(Loc[i, j])$ is the securely encrypted location, and $Enc_K(q_i)$ is homomorphic encryption of query component q_i. Finally, x $(1 \leq x < n)$ is the number of locations, where n is the total number of web links.

(a) Server 1 stores links as vectors. For instance, consider the link W_3:

$$W_3 = [0, 0, 3, 0, 5]$$

(b) The server homomorphically subtracts the encrypted query vector from W_3:

$$Enc_K(q' - W_3) = [Enc_K(3), Enc_K(3), Enc_K(0), Enc_K(3), Enc_K(-2)]$$

(c) The server homomorphically multiplies the resulting encrypted vector:

$$Enc_K\,(3 \times 3 \times 0 \times 3 \times (-2)) = Enc_K(0)$$

(d) The encrypted result is sent to the server 2 with encrypted location (say $Enc_{K'}(Loc[1, 3])$).

3.3. Server 2: Retrieval of relevant web links locations

After receiving encrypted computation results and corresponding encrypted locations from Server 1, Server 2

decrypts the computation results using key K (as shown in Algorithm 2). A decryption result of zero indicates that web link corresponding to query q is present on Server 1, whereas a non-zero result indicates its absence. If decryption result is zero, Server 2 forwards corresponding encrypted location to client for the final retrieval step.

To prevent access pattern leakage, computation results and their corresponding encrypted locations from Server 1 are stored in a randomized order on Server 2, rather than preserving their original sequence. Sequential storage could enable an adversary to correlate decrypted results with specific locations in Server 1, thereby inferring client's target. Random placement disrupts this correspondence and strengthens the privacy guarantees of the system. The following toy example illustrates these observations.

Example 2 (Based on Server 2 with Random Placement)

1. **Random Placement:** The encrypted computation results and corresponding encrypted locations received from Server 1 are stored in Server 2 in a randomized order rather than sequentially. For instance, if Server 1 outputs:

$$\big(Enc_K(0), Enc_{K'}(Loc[1,3])\big), \big(Enc_K(5), Enc_{K'}(Loc[1,4])\big), \big(Enc_K(4), Enc_{K'}(Loc[1,2])\big)$$

   they may be stored on Server 2 as:

$$\big(Enc_K(5), Enc_{K'}(Loc[1,4])\big), \big(Enc_K(4), Enc_{K'}(Loc[1,2])\big), \big(Enc_K(0), Enc_{K'}(Loc[1,3])\big)$$

   This prevents an adversary from inferring that first decrypted zero always corresponds to first link in Server 1.
2. **Server Side:**
   (a) Server 2 decrypts each computation result:

$$Dec_K(Enc_K(0)) = 0$$

   (b) For each zero result, Server 2 sends corresponding encrypted location (e.g., $Enc_{K'}(Loc[1,3])$) to client.
3. **Client Side:**
   (a) The client decrypts the received encrypted location:

$$Dec_{K'}(Enc_{K'}(Loc[1,3])) = Loc[1,3]$$

3.4. Server 3: Relevant links retrieval

This section helps retrieve the relevant web links. First, note that all links are stored on Server 3. Since web links can vary in size, storing them randomly may lead to inefficient memory utilization. To address this issue, we propose a novel restricted bin packing algorithm (RBA) (as shown in Algorithm 3) to minimize memory wastage. The main motivation of RBA is to divide each bin (where a bin is treated as a memory cell used to store web links) into a fixed number of slots, allowing links to be placed according to their respective weights (that is, the size of each web link). The entries stored in a bin (memory cell) are treated as a vector, which facilitates efficient homomorphic computations. In retrieval phase, client first encrypts the query vector consisting of encryptions of 0's, except at the desired location where the web link is present, represented by encryption of 1. This encrypted query vector is sent to the server. The server then performs homomorphic multiplication between the query vector and the bin (vector), followed by homomorphic addition. The resulting ciphertext is returned to the client, who decrypts it to obtain the desired web link(s) (as shown in Algorithm 4). A pictorial representation of this process is illustrated in Figure 3, and a toy example further demonstrates the procedure.

Algorithm 2 Server 2: Decryption and Location Retrieval

Input: Homomorphic results $\{r_i\}$, decryption key K, encrypted locations $\{\text{Enc}_{K'}(Loc[i,j])\}$
Output: Decrypted location(s) ($Loc[i,j]$) if query q is present
Server 2 Side:
Decrypt the results: $\{\text{Dec}_K(r_i)\}$
for each decrypted result $\text{Dec}_K(r_i)$ **do**
  if $\text{Dec}_K(r_i) = 0$ **then**
    Output: Query q is present; send corresponding encrypted location $\text{Enc}_{K'}(Loc[i,j])$ to the client
  end if
  if No $\text{Dec}_K(r_i) = 0$ found **then**
    Output: `abort` {Query not present; restart with new query}
  end if
end for
Client Side:
Decrypt the locations: $\{\text{Dec}(\text{Enc}_{K'}(Loc[i,j]))\}$, that is, $Loc[i,j]$

Example 3 (Based on Server 3)

1. **Client Side:**
   (a) The client encrypts the query vector (as the desired link is present at third slot of first bin):

$$q' = [0,0,1,0]$$

   where the total number of slots in each bin is four.
   (b) The client encrypts the query vector:

$$Enc_K(q') = [Enc_K(0), Enc_K(0), Enc_K(1), Enc_K(0)]$$

   (c) The encrypted query vector is sent to the server.
2. **Server Side:**

Algorithm 3 Restricted Bin Packing Algorithm (RBA)

Input: Set of n links with their weights $\gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$.
Output: Minimum number of bins required.
Step 1: Initialize
$Max_link_size \leftarrow \max(\gamma)$
$Min_link_size \leftarrow \min(\gamma)$
Step 2: Compute bin capacity
$Bin_capacity \leftarrow Max_link_size + \lceil \frac{Min_link_size}{2} \rceil$
Step 3: Compute possible number of slots in each bin
$Slots_per_bin \leftarrow \left\lfloor \frac{Max_link_size}{Min_link_size} \right\rfloor$
Step 4: Initialize empty bins and current bin capacity
$bins \leftarrow \emptyset$ {List of bins}
$current_bin_capacity \leftarrow Bin_capacity$
Step 5: For each item $\gamma_i \in \gamma$:
for each item $\gamma_i \in \gamma$ **do**
  if $\gamma_i < current_bin_capacity$ **then**
    Place γ_i in the current bin
    Update $current_bin_capacity \leftarrow current_bin_capacity - \gamma_i$
  else if $\gamma_i = current_bin_capacity$ **then**
    Start a new bin and place γ_i in the new bin
    Reset $current_bin_capacity \leftarrow Bin_capacity - \gamma_i$
  else
    Start a new bin
    Place γ_i in the new bin
    Reset $current_bin_capacity \leftarrow Bin_capacity - \gamma_i$
  end if
end for
Step 6: Handle incomplete bins
if $current_bin_capacity > 0$ **then**
  Divide the remaining capacity into R_s random slots
  $Total_slots_in_bin \leftarrow R_s +$ Number of occupied slots
  Fill the empty slots with the number 0
end if
Step 7: Output the number of bins used
Output $\#bins$

(a) The first bin contains the following items:

$$\text{Bin 1: } W_1, W_2, W_3, 0$$

(b) The server performs homomorphic multiplication of the query vector with each item in the bin:

$$[Enc_K(0 \times W_1), Enc(0 \times W_2), Enc(1 \times W_3), Enc(0 \times 0)]$$

which results in:

$$[Enc_K(0), Enc_K(0), Enc_K(W_3), Enc_K(0)]$$

(c) The server then performs homomorphic addition of the results:

$$Enc_K(0 + 0 + W_3 + 0) = Enc_K(W_3)$$

Algorithm 4 Web Link Retrieval Algorithm

Input: Location of web link $Loc[i, j]$.
Output: Web link corresponding to location $Loc[i, j]$.
Procedure:
Client Side:
Create a query vector q' with 0's except at the desired location $Loc[i, j]$, where it is set to 1.
Encrypt the query vector $\text{Enc}_K(q')$.
Send $\text{Enc}_K(q')$ to the server.
Server Side:
Divide the bins as per the Restricted Bin Packing Algorithm (RBA) to store web links.
for each bin b_i that stores web links **do**
  Compute $d_i = \text{Enc}_K(q') \cdot b_i$. {Homomorphic multiplication of the query vector with the bin.}
end for
Perform homomorphic addition on the results: $r = \sum d_i$. {Homomorphic addition of the multiplied results.}
Send the final result $\text{Enc}_K(r)$ back to the client.
Client Side:
Decrypt $\text{Enc}_K(r)$ to obtain the web link.
Output: Retrieve the desired web link.

(d) The encrypted result $Enc_K(W_3)$ is sent back to the client.

3. **Client Side:**
   (a) The client decrypts the received result:

$$Dec(Enc_K(W_3)) = W_3$$

   (b) The client retrieves W_3.

To further explain the design choices of proposed RBA, we provide its underlying rationale below.

3.4.1. Design Rationale of RBA for Web Link Storage

In this work, web links are modeled as items with varying sizes, where the size corresponds to the memory required to store each link. Unlike classical bin packing, where items are packed solely based on capacity constraints, our setting introduces an additional structural requirement: homomorphic retrieval over packed data.
Let $\gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ denote the sizes of web links. The objective is not only to minimize the number of bins but also to ensure that each bin supports vectorized homomorphic operations. This imposes the following constraints:

- Each bin must be representable as a fixed-length vector (for SIMD operations).
- The number of slots per bin must be bounded to allow efficient homomorphic multiplication.

The proposed Restricted Bin Packing Algorithm (RBA) enforces these constraints by defining:

1. A bounded bin capacity:

$$Bin_capacity = \max(\gamma) + \left\lceil \frac{\min(\gamma)}{2} \right\rceil$$

2. A fixed number of slots per bin:

$$Slots_per_bin = \left\lceil \frac{\max(\gamma)}{\min(\gamma)} \right\rceil$$

Lemma 1. The RBA ensures that each bin can be represented as a fixed-length vector without exceeding bin capacity.

Proof. Since each item $\gamma_i \leq \max(\gamma)$ and the total bin capacity is strictly greater than $\max(\gamma)$, at least one item always fits in a bin. Furthermore, the number of slots is upper bounded by the ratio $\max(\gamma)/\min(\gamma)$, ensuring that even the smallest items can be accommodated without overflow. Therefore, each bin can be safely represented as a fixed-size vector suitable for homomorphic operations. □

Unlike classical heuristics (that is, First-Fit or Best-Fit), RBA explicitly balances bin capacity and slot uniformity, which is essential for efficient encrypted computation. Hence, RBA is particularly suited for organizing web links in privacy-preserving retrieval systems.

Time Complexity Analysis of RBA. The Restricted Bin Packing Algorithm processes each of the n items exactly once, assigning each item to a bin based on the remaining capacity. Therefore, the overall time complexity of the algorithm is $O(n)$.

This is comparable to classical greedy bin packing algorithms such as First-Fit, which also operate in linear time when implemented efficiently [28]. The additional steps in RBA, including slot allocation and padding, introduce only constant-time overhead per bin and do not affect the overall asymptotic complexity. To further demonstrate the practicality of proposed framework, we analyze its scalability with respect to the number of categories.

3.5. Scalability with Large Number of Categories

Although the illustrative examples in this work consider five categories (L, G, B, T, Q) for clarity of presentation, the proposed framework naturally generalizes to a larger number of categories.

Let k denote the total number of categories. The client constructs a query vector $q' \in \mathbb{Z}^k$, and Server 1 stores category vectors of the same dimension. All homomorphic operations (subtraction and multiplication) are performed component-wise and therefore scale linearly with k. The key observations are as follows:

- The dimension of ciphertext vectors increases linearly with the number of categories.
- BFV batching (SIMD) allows multiple category components to be processed in parallel within a single ciphertext.
- The RBA based storage at Server 3 remains independent of category count, as it depends only on link sizes.

Therefore, the proposed system can scale to hundreds or thousands of categories, with the primary overhead being linear growth in vector dimension, which is manageable under modern homomorphic encryption parameters.

Category	Average Execution Time (sec)
L	240.7001
G	210.7380
B	210.7510
T	250.0000
Q	260.5605

Table 2
Average execution time for Server 1 computation across categories.

4. Implementation and Performance Evaluation

In this section, we describe the implementation setup and its results. We then compare our protocol with the baseline protocol, Coeus [13], which retrieves the desired web link in less time while performing fewer FHE operations and requiring less client side storage.

4.1. Implementation Setup

We implemented both protocols using Jupyter Notebook on Windows 11 with Python (version 3.9.10) and TenSEAL library [15]. The experiments were conducted on a machine equipped with an Intel(R) Core(TM) i5-10500 CPU@3.10GHz and 16GB of memory. These specifications were sufficient to evaluate the performance of our PIR framework.

Homomorphic encryption details: Both protocols were implemented using BFV scheme in TenSEAL library, which supports exact arithmetic on encrypted integers. The default parameters were used, including a plaintext modulus of $2^{16}+1$ and a ring dimension of 2^{13}.

Data collection: We collected web links from Wikipedia, resulting in a total of 34,930 links.[2] The distribution of links are as follows: 16,365 for category L; 26,111 for category G; 10,898 for category B; 11,781 for category T; and 10,610 for category Q.[3]

4.2. Implementation of Our Protocol

In this subsection, we present the implementation results of our protocol. Server 1 stores the metadata, Server 2 stores the computation results derived from Server 1, and Server 3 stores the web links according to RBA. We executed our program twice and report the average execution times for initialization of Server 1 and Server 3 as 29.13 seconds and 47.29 seconds, respectively. Table 2 presents the computation time for each category, while Table 3 summarizes the outcome of restricted bin packing algorithm.

[2] A single link may belong to multiple categories.

[3] The dataset is publicly available at https://github.com/akshit-aggarwal/Empowering-Privacy-A-Zero-Cost-Protocol-for-Concealing-LGBTQ-Search-Queries.

Specification	Value
Total number of web links	34,930
Total number of bins	18,854
Number of slots per bin	9

Table 3
Outcome of restricted bin packing algorithm.

# Retrieval	Server 2 Execution Time (sec)				
	L	G	B	T	Q
1	14.93	0.02	0.02	17.16	15.43
10	16.85	0.25	0.26	17.25	16.86
100	18.45	2.34	1.50	18.55	19.58
1000	33.11	16.14	16.86	33.80	34.46

Table 4
Server 2 execution time for metadata retrieval across categories.

# Retrieval	Server 3 Execution Time (sec)				
	L	G	B	T	Q
1	0.0241	0.0216	0.0216	0.0216	0.0290
10	0.2198	0.1711	0.1463	0.1749	0.1746
100	0.4067	0.3695	0.4351	0.3742	0.4284
1000	3.0884	2.6792	2.6519	4.0001	3.7355

Table 5
Server 3 execution time for web link(s) retrieval across categories.

# Retrieval	Total Average Execution Time (sec)				
	L	G	B	T	Q
1	254.9541	210.7796	210.7927	267.1816	276.0195
10	257.0718	211.0591	211.0493	267.4249	277.5951
100	258.8567	212.9477	212.6905	268.9242	280.5099
1000	276.1984	229.5572	230.2530	287.8001	298.7560

Table 6
Total average execution time for link retrieval across categories.

Next, we executed our protocol for multiple iterations and measured the metadata retrieval time (Table 4) and web link(s) retrieval time (Table 5). The total average execution times are summarized in Table 6.

The execution times reported in Tables 4 and 5 appear close due to deterministic nature of homomorphic operations and controlled experimental environment. Unlike traditional systems where variability arises from input/output or network fluctuations, our implementation primarily involves CPU bound cryptographic operations, which exhibit low variance across runs.

To ensure reliability, each experiment was executed multiple times and averaged. The small variation indicates stability of the system rather than lack of significance. Furthermore, the consistent gap between our protocol and the baseline protocol demonstrates meaningful performance improvement.

Overall, from Tables 4-6, we observe that execution time increases with the number of links due to sequential processing of encrypted data. This behavior is expected in PIR systems, where traditional indexing techniques cannot be directly applied without compromising privacy.

However, this does not limit the scalability of the proposed system. In particular:

- The computation can be parallelized across multiple cores or distributed servers.
- Homomorphic batching (SIMD) enables processing of multiple data elements within a single ciphertext.
- The proposed three server architecture naturally supports horizontal scaling in cloud environments.

Therefore, while the current implementation exhibits linear growth, the system remains suitable for large scale settings through parallel and distributed execution.

Category	Server 1 Average Execution Time (sec)
L	1133.1100
G	1093.8700
B	1011.2500
T	1018.2500
Q	1018.3750

Table 7
Average execution time for Server 1 computation across categories in Coeus protocol.

# Retrieval	Server 2 Execution Time (sec)				
	L	G	B	T	Q
1	0.1000	0.1000	0.1000	0.1000	0.1165
10	0.9335	0.9000	0.9335	0.8835	0.9165
100	8.8335	8.8165	8.8335	8.8165	8.8335
1000	88.3835	88.0500	87.8355	88.0835	88.0000

Table 8
Server 2 execution time for metadata retrieval in Coeus protocol across number of categories.

4.3. Implementation of the Baseline Protocol

In this subsection, we present the implementation results of the baseline protocol, Coeus [13]. This protocol is divided into three servers: Server 1 stores metadata, Server 2 stores link locations, and Server 3 stores web links using bin packing algorithm. We executed the program twice, and the average execution times for initialization of Server 1, Server 2, and Server 3 are 29.12 seconds, 1.16 seconds, and 15.75 seconds, respectively. Tables 7-9 present data retrieval times from each server, while Table 10 reports the total average execution time.

From Tables 7–10, we observe that execution time increases with the number of links due to linear search. Similar to our protocol, Coeus enables clients to securely retrieve the desired web links without revealing query information.

# Retrieval	Server 3 Execution Time (sec)				
	L	G	B	T	Q
1	0.0043	0.0010	0.0012	0.0021	0.0024
10	0.0100	0.0095	0.0102	0.0102	0.0101
100	0.1243	0.1209	0.1084	0.0932	0.1100
1000	0.9089	0.8920	0.9341	0.8721	0.8954

Table 9
Server 3 execution time for link retrieval in Coeus protocol across categories.

# Retrieval	Total Average Execution Time (sec)				
	L	G	B	T	Q
1	1133.2143	1093.9710	1011.3512	1018.2221	1018.4939
10	1134.0535	1094.7795	1012.1937	1019.0137	1019.3016
100	1142.0678	1102.8074	1020.0288	1027.0288	1027.3185
1000	1222.0689	1182.8120	1100.0196	1107.0756	1107.2704

Table 10
Total average execution time for web link(s) retrieval in Coeus protocol across categories.

4.4. Evaluation Metrics

In this subsection, we evaluate the performance of our proposed protocol and compare it with the baseline Coeus protocol using three key metrics (i) execution time, (ii) number of homomorphic (FHE) operations, and (iii) client-side storage requirements. These metrics collectively capture the computational efficiency, cryptographic cost, and storage overhead of the protocols.

4.4.1. Execution time

Our results show that the client requires significantly less execution time compared to baseline protocol (as shown in Figure 4). Across five categories, our protocol achieves 4.42× speedup for category L, 5.18× for G, 4.79× for B, 3.82× for T, and 3.68× for Q. Overall, this corresponds to an average 4.38× improvement in execution time compared to baseline protocol.
The observed performance improvement primarily stems from reduced homomorphic operations on Server 1 and the elimination of client side post processing and storage overhead.

4.4.2. Number of FHE operations

Our protocol requires only one vector multiplication and 34,930 vector subtractions for a single web link retrieval, whereas the baseline protocol requires 34,931 vector multiplications. Since homomorphic multiplication is significantly more expensive than homomorphic subtraction, this reduction in the number of multiplications directly contributes to the improved execution time observed in our protocol.

4.4.3. Client-side storage

We now evaluate client-side storage requirements for single web link retrieval. Table 11 reports the amount of storage required at client side for both our protocol and the baseline Coeus protocol across different categories.

Category	Client-side Storage (MB)	
	Our Protocol	Coeus Protocol
L	0.001226	3.485133
G	0.138528	3.523303
B	0.138528	3.523303
T	0.001226	3.485133
Q	0.001226	3.485133

Table 11
Comparison of client-side storage requirements between our protocol and Coeus protocol.

We remark that the Client-side storage is measured as the total size of ciphertexts received and stored during query execution. In baseline protocol, the client receives multiple encrypted results corresponding to all candidate links, leading to storage proportional to the number of links (approximately 3.5 MB in our experiments).
In contrast, the proposed protocol returns only the encrypted location identifiers corresponding to matching results. Since only a small number of locations are retrieved, the storage requirement is significantly reduced. Thus, the reduction in storage is achieved by:

- Avoiding bulk transfer of encrypted data,
- Returning only relevant encrypted locations,
- Eliminating client-side post-processing.

From Table 11, the proposed protocol significantly reduces client side storage across all categories. This reduction is achieved by avoiding bulk transfer of encrypted data and returning only the relevant encrypted locations, thereby eliminating unnecessary client side storage and post-processing. In summary, the proposed protocol achieves exact query retrieval with significantly lower execution time, fewer homomorphic operations, and substantially reduced client-side storage requirements compared to baseline Coeus protocol.

4.4.4. Comparison with Other PIR Schemes

In addition to the Coeus protocol, several modern PIR schemes have been proposed in the literature, including single server FHE based approaches [12, 20] and multi-server constructions [14].
Single server FHE based PIR schemes achieve strong privacy guarantees but suffer from high computational overhead due to a large number of homomorphic multiplications. In contrast, our protocol significantly reduces the number of multiplications by replacing them with cheaper subtraction operations. Whereas multi-server PIR schemes improve efficiency but often rely on additional assumptions such as data replication or incur communication overhead. Our approach differs by:

- Eliminating client side storage overhead,
- Reducing homomorphic multiplications,

- Avoiding oblivious RAM.

A detailed experimental comparison with all such schemes is left for future work due to differences in system assumptions and implementation environments.

5. Security Analysis and Correctness

This section presents the security and correctness analysis of the proposed protocol. We first analyze its security under the considered adversarial model, followed by correctness analysis. We also briefly discuss the impact of potential server collusion and possible mitigation strategies.

5.1. Security Analysis

This subsection presents a formal security analysis of the proposed protocol. We begin by defining adversarial model and associated security goals. We then prove that the protocol achieves query privacy against an honest-but-curious adversary under semantic security of BFV homomorphic encryption scheme.

5.1.1. Threat Model

We assume that the adversary may access the data stored on servers, observe and record all client queries and responses, and perform any form of analysis on them. The adversary is assumed to be honest but curious, meaning it does not tamper with server side computations or corrupt the data stored on the servers. However, such an adversary may still generate incorrect responses, thereby violating integrity. We also assume that the adversary cannot break standard cryptographic assumptions, such as semantic security of the encryption scheme.

5.1.2. Goal

Our protocol must guarantee query privacy for its clients. An adversary should not be able to infer any information about client's query q. This also implies that the adversary must not learn any information about the data retrieved.

5.1.3. Security Proof

Let us define a security game $\mathcal{G}_0$ between a challenger and an adversary. The adversary supplies two queries q^0 and q^1 of the same size, and challenger randomly selects one of them as q^α, where $\alpha \in \{0, 1\}$. The challenger performs a query with q^α using our protocol. Then, the adversary outputs its guess $\alpha' \in \{0, 1\}$. The adversary wins the game if its guess is correct, that is, $\alpha' = \alpha$ in $\mathcal{G}_0$.
Let E_0 be the event that $\alpha' = \alpha$ in $\mathcal{G}_0$. Let us also consider another game $\mathcal{G}_1$ where challenger simulates our protocol with a query selected uniformly at random from query space, and let E_1 be the event that $\alpha' = \alpha$ in $\mathcal{G}_1$. Evidently, $\Pr[E_1] = 1/2$, since the challenger's query is independent of those provided by the adversary, and adversary can only perform random guesses. Now, in $\mathcal{G}_0$, according to our protocol, challenger encrypts q^α using BFV homomorphic encryption scheme. Therefore, the advantage of the adversary being able to distinguish it from a random string is given by

$$\Big|\Pr[E_0] - \Pr[E_1]\Big| \leq \epsilon_{\mathsf{BFV}}.$$

Substituting the value of $\Pr[E_1]$, we get

$$\Big|\Pr[E_0] - 1/2\Big| \leq \epsilon_{\mathsf{BFV}}.$$

Thus, the adversary cannot win the game $\mathcal{G}_0$ with non-negligible advantage.
We now discuss an additional security consideration related to potential server collusion.

5.2. Collusion Resistance

The security of the proposed protocol relies on standard non-collusion assumption among servers. If Server 1 and Server 2 collude, they may combine homomorphic computation results with corresponding decrypted outputs, potentially revealing query information. To mitigate such risks, the following approaches can be adopted:

- Introducing additional non-colluding servers to distribute trust,
- Employing secure multi-party computation techniques,
- Using threshold homomorphic encryption, where no single server can decrypt independently,
- Incorporating trusted hardware (say, secure enclaves) to isolate computations.

Handling collusion-resistant PIR remains an important direction for future work.

5.3. Correctness Analysis

Let the dataset consist of n publicly available web links $\{W_i\}_{i=1}^n$, where each link W_i is associated with a category identifier $c_i \in \{1, 2, 3, 4, 5\}$ corresponding to the categories $\{\mathrm{L}, \mathrm{G}, \mathrm{B}, \mathrm{T}, \mathrm{Q}\}$. Let the client query be $q \in \{1, 2, 3, 4, 5\}$.
To enable homomorphic equality testing, the client performs a query expansion and constructs an encrypted query vector

$$\mathrm{Enc}(q') = \mathrm{Enc}(q, q, \ldots, q) \in \mathbb{Z}^m,$$

where m denotes the expansion length determined by the system parameters. Server 1 stores, for each link W_i, an expanded category vector

$$c_i' = (c_i, c_i, \ldots, c_i) \in \mathbb{Z}^m.$$

Server 1 homomorphically computes the component-wise difference

$$\mathrm{Enc}(d_i) = \mathrm{Enc}(q' - c_i'),$$

and then evaluates the homomorphic product

$$\mathrm{Enc}(r_i) = \mathrm{Enc}\left(\prod_{j=1}^{m}(q_j' - c_{i,j}')\right).$$

By construction, $r_i = 0$ if and only if $q = c_i$, and $r_i \neq 0$ otherwise. Server 1 forwards the ciphertexts $\text{Enc}(r_i)$ along with the corresponding encrypted location identifiers to Server 2 in a randomly permuted order.
Server 2 decrypts $\text{Enc}(r_i)$ and forwards only those encrypted location identifiers for which $r_i = 0$, thereby preserving access pattern privacy through random placement.
The client receives the encrypted locations and constructs an encrypted selection vector $\text{Enc}(s)$ such that $s_{i^\star} = 1$ for each matching index $i^\star$ and $s_i = 0$ for all other indices. Server 3 homomorphically computes

$$\text{Enc}(R) = \sum_{i=1}^{n} \text{Enc}(s_i) \cdot \text{Enc}(W_i),$$

and returns the result to the client.

Lemma 1 (Correctness). If there exists an index $i^\star$ such that $c_{i^\star} = q$, then the client decrypts $\text{Enc}(R)$ to obtain the corresponding web link $W_{i^\star}$.

Proof. For all $i \neq i^\star$, we have $q - c_i \neq 0$, which implies $r_i \neq 0$ and hence $s_i = 0$. For $i = i^\star$, $q - c_{i^\star} = 0$, yielding $r_i^\star = 0$ and $s_{i^\star} = 1$. Therefore,

$$\text{Enc}(R) = \text{Enc}(R_{i^\star}),$$

and correct decryption follows from the correctness property of the BFV homomorphic encryption scheme. □

6. Limitations

The proposed work does not verify the designed protocol concerning side-channel attacks (or any cryptographic attacks). The proposed work currently does not retrieve the results for more than one query at one time. However, it can retrieve results for more than one query asynchronously.

7. Conclusion

In this work, we introduced a novel PIR protocol based on three server architecture. By strategically distributing computational load, our protocol achieves a significant reduction in the number of homomorphic operations and requires substantially reduced client side storage. Furthermore, our protocol preserves the same level of query privacy (i.e., zero leakage) and correctness as the baseline protocol, while significantly improving efficiency. Experimental results demonstrate an average 4.38× speedup over the baseline protocol. Hence, our protocol enables clients to securely and efficiently retrieve relevant data from any large database.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Data availability statement

The data that support the findings of this study are available from the corresponding author (Akshit Aggarwal) on request

Funding

No funding was received to support this research.

References

[1] J. Allen, "Recent cyber attacks & data breaches in 2021." https://purplesec.us/recent-cyber-security-attacks/, 2021. Accessed: 2022-12-14.

[2] I. Vojinovic, "Data breach statistics that will make you think twice before filling out an online form." https://dataprot.net/statistics/data-breach-statistics/, 2022. Accessed: 2023-07-23.

[3] B. Chor and N. Gilboa, "Computationally Private Information Retrieval," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 304–313, 1997.

[4] D. Demmler, P. Rindal, M. Rosulek, and N. Trieu, "PIR-PSI: Scaling Private Contact Discovery," *Proceedings on Privacy Enhancing Technologies*, p. 159–178, 2018.

[5] S. Angel and S. Setty, "Unobservable Communication over Fully Untrusted Infrastructure," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 551–569, 2016.

[6] A. Aggarwal, "A Tree-Based FHE Approach for Private Set Intersection," *Journal of Advanced Research in Applied Sciences and Engineering Technology*, vol. 58, no. 5, pp. 27–38, 2026.

[7] A. Singh and K. Chatterjee, "SecEVS: Secure Electronic Voting System using Blockchain Technology," in *Proceedings of the International Conference on Computing, Power and Communication Technologies (GUCON)*, pp. 863–867, 2018.

[8] A. Vadapalli, F. Bayatbabolghani, and R. Henry, "You May Also Like... Privacy: Recommendation Systems Meet PIR," *Proceedings on Privacy Enhancing Technologies*, pp. 30–53, 2021.

[9] A. Aggarwal, "Ring Ring! Who's There? A Privacy Preserving Mobile Number Search." Cryptology ePrint Archive, Paper 2024/2020, 2024.

[10] R. Henry, "Polynomial Batch Codes for Efficient IT-PIR," *Cryptology ePrint Archive*, 2016.

[11] D. Kogan and H. Corrigan-Gibbs, "Private Blocklist Lookups with Checklist," in *30th USENIX security symposium (USENIX Security 21)*, pp. 875–892, 2021.

[12] I. Ahmad, D. Agrawal, A. E. Abbadi, and T. Gupta, "Pantheon: Private Retrieval from Public Key-Value Store," *Proceedings of the VLDB Endowment*, vol. 16, no. 4, pp. 643–656, 2022.

[13] I. Ahmad, L. Sarker, D. Agrawal, A. El Abbadi, and T. Gupta, "Coeus: A System for Oblivious Document Ranking and Retrieval," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pp. 672–690, 2021.

[14] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan, "Single-Server Private Information Retrieval with Sublinear Amortized Time," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 3–33, 2022.

[15] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, "TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption," *arXiv preprint arXiv:2104.03152*, 2021.

[16] 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*, pp. 364–373, 1997.

[17] I. Goldberg, "Improving the Robustness of Private Information Retrieval," in *IEEE Symposium on Security and Privacy (SP'07)*, pp. 131–148, 2007.

[18] X. Yi, M. G. Kaosar, R. Paulet, and E. Bertino, "Single-Database Private Information Retrieval From Fully Homomorphic Encryption," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 1125–1134, 2012.

[19] S. Angel, H. Chen, K. Laine, and S. Setty, "PIR with Compressed Queries and Amortized Query Processing," in *IEEE Symposium on Security and Privacy (SP)*, pp. 962–979, 2018.

[20] J. Fu, N. Wang, B. Cui, and B. K. Bhargava, "A Practical Framework for Secure Document Retrieval in Encrypted Cloud File Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 5, pp. 1246–1261, 2021.

[21] A. Aggarwal, Y. Li, and S. Swain, "Private LGBTQ Searches Without a Trace," in *International Conference on Applied Cryptography and Network Security*, pp. 213–217, Springer, 2025.

[22] S. Colombo, K. Nikitin, H. Corrigan-Gibbs, D. J. Wu, and B. Ford, "Authenticated Private Information Retrieval," in *32nd USENIX security symposium (USENIX Security 23)*, pp. 3835–3851, 2023.

[23] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure kNN Computation on Encrypted Databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp. 139–152, 2009.

[24] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, "One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval," in *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 3889–3905, 2023.

[25] A. Davidson, G. Pestana, and S. Celi, "FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval," in *Proceedings on Privacy Enhancing Technologies*, pp. 365–383, 2023.

[26] M. Zhou, A. Park, E. Shi, and W. Zheng, "Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation," *Cryptology ePrint Archive*, 2023.

[27] O. Regev, "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[28] E. G. Coffman, J. Csirik, G. Galambos, S. Martello, D. Vigo, *et al.*, "Bin Packing Approximation Algorithms: Survey and Classification.," in *Handbook of combinatorial optimization*, pp. 455–531, 2013.

[29] P. FIPS, "197. Advanced Encryption Standard (AES), National Institute of Standards and Technology, US Department of Commerce," 2001.

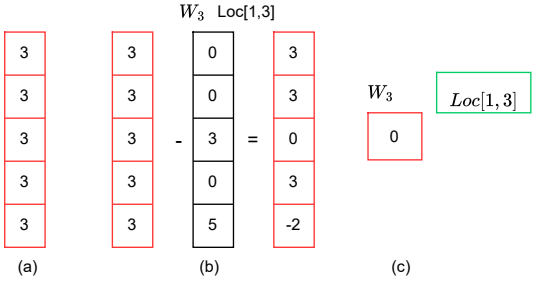

Figure 2: "Red" denotes vectors encrypted via homomorphic encryption, "Green" denotes those encrypted using a secure mechanism, and "Black" denotes plaintext vectors. (a) shows encrypted query expansion, (b) homomorphic subtraction, and (c) homomorphic multiplication of the results from (b).

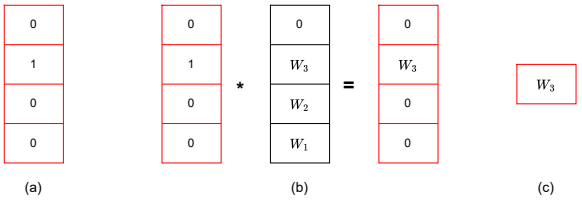

Figure 3: "Red" color represents encrypted vector, and "Black" color represents plaintext vector. (a) indicates an expansion of encrypted query (where the desired location contains 1 and other contains 0's). (b) indicates homomorphic multiplication. (c) indicates homomorphic addition of (b)'s results.

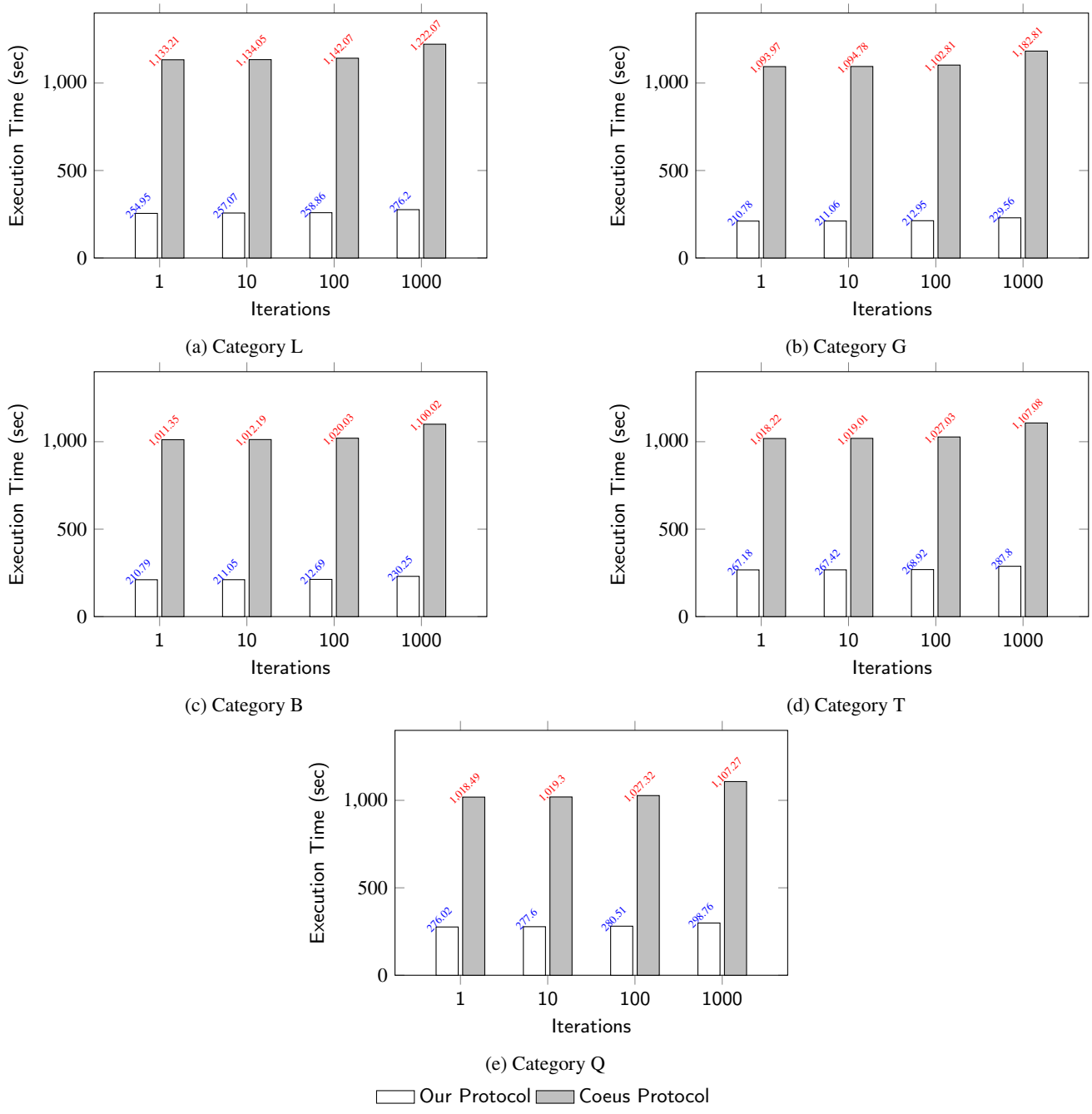

Figure 4: Comparison of Execution Times between Our Protocol and Coeus Protocol across categories (L, G, B, T, Q).