Two-Server Offline/Online Private Information Retrieval With Small Client Storage

Cheng Huang, *Member, IEEE*, Anjia Yang, *Member, IEEE*, Dongxiao Liu, *Member, IEEE*, Rongxing Lu, *Fellow, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—In this paper, we propose PIRS, a two-server offline/online private information retrieval scheme with small client storage. In PIRS, a client first engages in an offline phase to preprocess a database replicated on two servers to generate query-independent hints. Utilizing the pre-computed hints, the client then securely retrieves any record from the database without exposing its index during an online phase, with the server-side computational complexity being sublinear for high efficiency. Compared to state-of-the-art schemes, PIRS distinguishes itself by enabling the client to outsource the hints to the servers instead of storing them, dramatically reducing the local storage requirement from GB/MB to MB/KB, given that the size of the hints is directly proportional to the database volume. Specifically, the client employs secret sharing to achieve secure hint outsourcing and only fetches the relevant hint for each online PIR query. In such an outsourcing environment, we introduce a new technique named oblivious switching to obfuscate repeated hint/record accesses, and carefully tailor online PIR queries to guarantee sublinear computational complexity. Furthermore, we propose a secure and efficient method that delegates the task of locating appropriate hints for particular PIR queries to the servers, thus avoiding costly computations or extra data storage on the client side. Finally, we conduct a comprehensive security analysis to demonstrate PIRS's security, and develop a proof-of-concept prototype to show the practicality of PIRS in terms of computational, communication, and storage overheads.**

***Index Terms*—Private information retrieval, offline/online, preprocessing, two servers, small client storage, outsourcing.**

I. Introduction

PRIVATE Information Retrieval (PIR) is an essential privacy protection tool for enhancing a spectrum of applications, including cloud computing/storage [1], data trading [2], messaging [3], advertising [4], certificate management [5], and video streaming [6]. Generally, PIR allows a client to retrieve a piece of data from a database in a privacy-preserving manner, i.e., the client can retrieve a single file from a large digital library without revealing which file is of interest. Modern PIR schemes can be roughly categorized based on the number of deployed servers: single-server models, where data is hosted on one server [7], [8], [9], [10], and two/multi-server models, with data replicated across two or multiple servers [11], [12], [13], [14], [15]. Single-server PIR often relies on partially or fully homomorphic encryption (PHE/FHE) that complies with strict security assumptions, with its efficiency hinging on PHE/FHE performance and novel designs. Differently, two/multi-server PIR operates under the assumption that not all servers are compromised or colluded with each other. The assumption is less stringent, but it enables the construction of notably efficient PIR schemes that have attracted considerable research attention.

Among the state-of-the-art works, two-server PIR schemes based on distributed point functions (DPF) stand out by their communication efficiency [16], [17], [18], [19]. The core advantage of DPF lies in its unique capability: it enables two servers to jointly evaluate a private function that is non-zero at a single point, while maintaining the confidentiality of that value during the evaluation. This feature can be utilized to realize two-server PIR with logarithmic communication efficiency. Although these schemes are impressive, they still incur linear computational costs on the server side inevitably. To overcome the limitation and improve the server's throughput, two-server offline/online PIR schemes (also known as stateful PIR schemes) are proposed [12], [13], [14], [20].

One of the groundbreaking two-server offline/online PIR schemes was proposed by Corrigan-Gibbs and Kogan [12]. In their scheme, given a database $\mathbb{DB}$ of size N, the client first generates hints by interacting with two servers during the offline phase. Each hint involves a subset $\mathbb{S}$ of $\sqrt{N}$ distinct random indices from the interval $[0, N-1]$ and an associated Parity Word $\oplus_{i\in\mathbb{S}}\mathbb{DB}[i]$, where $\mathbb{DB}[i]$ denotes the i-th database entry. The client then locally stores these hints. In the online phase, to retrieve the i-th record from the database, the client locates a subset $\mathbb{S}$ that includes i, requests from the servers the Parity Word of the subset $\mathbb{S}\backslash\{i\}$, and combine it with the Parity Word of the subset $\mathbb{S}$ to recover $\mathbb{DB}[i]$. It can be observed that the server only needs to touch $\sqrt{N}-1$ entries during the online phase, and the computational complexity is sublinear to the

Received 3 February 2024; revised 31 October 2025; accepted 10 December 2025. Date of publication 18 December 2025; date of current version 12 March 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62402115 and in part by the State Key Laboratory of Integrated Services Networks, Xidian University, under Grant ISN26-07. *(Corresponding authors: Cheng Huang; Anjia Yang.)*

Cheng Huang is with the College of Computer Science and Artificial Intelligence, Fudan University, Shanghai 200438, China, and also with State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China (e-mail: chuang@fudan.edu.cn).

Anjia Yang is with the College of Cyber Security, Jinan University, Guangzhou 510632, China, and also with Pazhou Lab, Guangzhou 510335, China (e-mail: anjiayang@gmail.com).

Dongxiao Liu is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: dongxiao.liu@uestc.edu.cn).

Rongxing Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada (e-mail: rlu1@unb.ca).

Xuemin Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: sshen@uwaterloo.ca).

Digital Object Identifier 10.1109/TDSC.2025.3645856

database size, which is $\mathcal{O}(\sqrt{N})$. Furthermore, they proposed a technique named puncturable pseudorandom sets (PPRS) to compress a subset into a PIR key, reducing the communication complexity from $\mathcal{O}(\sqrt{N})$ to $\mathcal{O}(\lambda \cdot \log_2 \sqrt{N})$, where λ is a security parameter, and also proposed a hint refreshing approach to prevent privacy leakages caused by hint reusing [13]. Recently, several promising PIR schemes have been established based on [12], [13]. Some works focus on extending the two-server offline/online PIR schemes to single-server models by utilizing PHE/FHE or streamlining offline preprocessing [21], [22], while others discuss how to support mutable databases, considering that databases may be changed and updated over time [13], [14].

Although previous two-server offline/online PIR schemes hold potential for a wide range of application scenarios, they all encounter a serious concern: the client storage for hints escalates linearly with the size of the database and its entries. For example, with a database comprising $10,000$ records, each of 100 KB size, the client needs to store at least 1 GB of hints for online PIR. Such storage demands may become untenable for a client with limited storage resources. Therefore, minimizing client storage costs while preserving sublinear online computational complexity on the server side is desirable. To address the issue, a natural idea is to outsource the client's hints to two servers via secret sharing, and then allow the client to fetch the necessary hint for each online PIR query as needed. Nevertheless, designing such a new offline/online PIR scheme is far from trivial, as the behavior of retrieving one particular hint leaks the index of that hint within the entire outsourced hints, and leads to security issues. An obvious problem with outsourcing hints is how to refresh a used hint in an outsourcing environment. If a client directly replaces a used hint with a refreshed one at the corresponding index, this action may inadvertently signal to the servers that identical hint indices indicate repeated access to the same database record. Such a pattern could compromise the PIR's privacy guarantees. To overcome the challenge, we propose a method named oblivious switching. This technique allows for the replacement of a used hint with an unused one randomly chosen from all outsourced hints in a manner that is oblivious to the servers, without modifying the hints' distributions. In other words, a used hint is refreshed only after its index has been obfuscated, ensuring that the servers have a negligible probability of deducing the index of the refreshed hint. Consequently, this helps to conceal the access pattern that reveals repeated access to the same database record based on identical hint indices. The oblivious switching technique primarily utilizes the 'obliviously write' characteristic of DPF, which enables a client to overwrite an outsourced hint without disclosing its index. Moreover, we carefully tailor online PIR queries to ensure that the client can obtain a random, non-used hint for substitution with the used hint.

In addition to the client storage concerns mentioned above, another issue with existing two-server offline/online PIR schemes is that PPRS, constructed from puncturable pseudorandom functions (PRFs), does not support fast membership testing due to the non-invertibility property. In contrast, PPRS constructed from pseudorandom permutations do support fast membership testing but at the cost of increased online communication overheads from $\mathcal{O}(\lambda \cdot \log_2 \sqrt{N})$ to $\mathcal{O}(\sqrt{N})$. Here, fast membership testing refers to the process of quickly locating the index of a hint associated with a given query index. Specifically, with PRF-based PPRS, finding a proper hint that matches a query index requires at most λN PRF operations. To circumvent the heavy computations, the client can store an additional data structure of size $\lambda N \cdot \log_2 N$ to expedite hint locating, as discussed in [12], [13]. Alternatively, our proposed solution is to delegate the task of hint locating to the servers. In this way, the client is relieved of the burden of conducting costly local computations or maintaining supplementary data structures, yet retains the ability to pinpoint the requisite hint efficiently through interaction with the servers. To achieve the task delegation, we introduce linearly homomorphic encryption with preprocessing (LHEP) and integrate it with bloom filters, super-increasing sequences, and Beaver's matrix multiplication techniques. Bloom filters are used to encode the hint before delegation. LHEP and super-increasing sequences are employed to enable secure and highly efficient hint locating on the server side by transforming the membership testing operations to dot product operations. However, LHEP may cause huge storage costs or online communication overheads; as a result, we also employ Beaver's matrix multiplication to streamline communication costs or local storage costs through offline preprocessing. In summary, the main contributions of the paper are threefold:

- First, we propose a new two-server offline/online PIR scheme, named PIRS, that can alleviate a client's storage burden, i.e., reducing the local storage costs from GB/MB to MB/KB, regardless of the database record size, while concurrently preserving sublinear online computational complexity.
- Second, we develop a secure hint locating method, which enables a client to efficiently find proper hints for any PIR queries by server interaction with small communication costs, avoiding linear computational costs and eliminating the need for additional storage.
- Third, we provide a simulation-based security analysis to demonstrate the security of PIRS, and we also develop a proof-of-concept prototype to show its practicality in terms of computation, communication, and storage overheads, with databases containing millions of records and record contents at the KB level.

The remainder of this paper is organized as follows. In Section II, we introduce the system model, define the security model, and identify the design goals. Then, we present preliminaries in Section III, and propose the two-server offline/online PIR scheme in Section IV. Subsequently, security analysis and performance evaluation are given in Sections V and VI, respectively. Finally, Section VII reviews some related works and Section VIII draws the conclusion.

II. Models and Design Goals

A. Notations

We introduce the following notations for clarity: Matrices are denoted by *bold uppercase letters*, e.g., $\mathbf{M}$. Vectors use *bold*

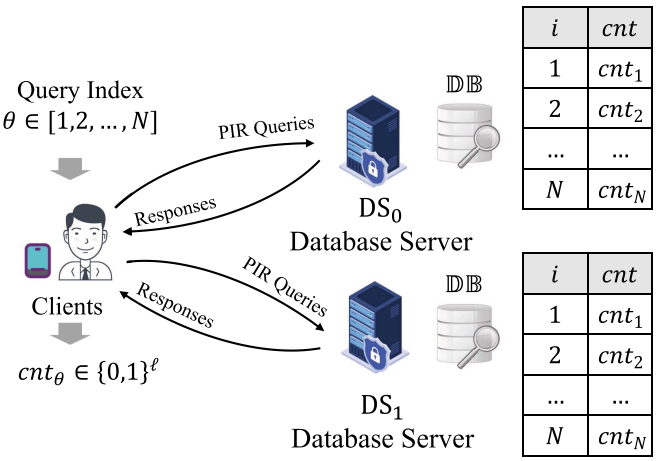


Fig. 1. System model.

lowercase letters, e.g., $\mathbf{v}$, and set entities are represented by *hollow uppercase letters*, e.g., $\mathbb{S}$. For matrices, $\mathbf{M}[i,:]$ denotes the i-th row, while $\mathbf{M}[:,j]$ represents the j-th column. For vectors and sets, $\mathbf{v}[i]$ and $\mathbb{S}[i]$ indicate the i-th element, respectively. The superscripts 0 and 1 indicate the secret sharing of these constructs. For example, secret shared forms of matrix $\mathbf{M}$ are $\mathbf{M}^0$ and $\mathbf{M}^1$, and for vector $\mathbf{v}$, they are $\mathbf{v}^0$ and $\mathbf{v}^1$. For symbols 0 or 1, an overline indicates negation; thus, for $\beta = 0$, $\bar{\beta}$ is 1, and for $\beta = 1$, $\bar{\beta}$ is 0.

B. System Model

A remote database system consists of two entities: a client and database servers. The servers are responsible for managing a database $\mathbb{DB} = (i, \mathsf{cnt}_i)_{i=1}^N$, which is represented by a collection of (index, content) pairs. Without loss of generality, we assume that the size of stored contents is standardized by padding smaller ones with dummy data or cutting bigger ones to data chunks, such as all contents have the same size, i.e., $|\mathsf{cnt}_i| = |\mathsf{cnt}_j| = \ell$ for any $i \neq j$ and $i, j \in [1, N]$.

Two-Server Model: In such a system, a client seeks to retrieve the i-th record cnt_i from $\mathbb{DB}$, without revealing the index i to the servers managing the database. Particularly, a two-server model is considered, where two servers, DS_0 and DS_1, are deployed with identical copies of $\mathbb{DB}$, as shown in Fig. 1.

C. Threat Model & Security Model

Threat Model: Database servers are usually held by cloud service providers that cannot afford loss of reputation and negative impacts associated with being caught cheating. Consequently, the servers are assumed to be *honest-but-curious*, i.e., they honestly provide database services, but may be curious about a client's queries and analyze the client's queries by launching passive attacks. Under such a threat model, other malicious attacks on availability (e.g., intentionally deleting stored records or rejecting any client's queries) are out of scope.

Trust Model: In the system, a client is trusted, and two database servers, DS_0 and DS_1, do not collude with each other. The assumption is reasonable to some extent as each server can be maintained by a separate cloud service provider. The non-collusion assumption has been widely adopted in recent works [23], [24], [25].

Syntax of Two-Server Offline/Online PIR: We define a two-server offline/online PIR scheme, Π, that consists of seven algorithms: $\mathtt{HtGen}(\cdot)$, $\mathtt{HtPrep}(\cdot)$, $\mathtt{Query}(\cdot)$, $\mathtt{Answer}(\cdot)$, $\mathtt{Recov}(\cdot)$, $\mathtt{HtUpd}(\cdot)$, and $\mathtt{HtRfsh}(\cdot)$:

- $\mathtt{HtGen}(\mathcal{M}, \mathbb{DB}) \to \mathbb{HT}$: The algorithm is executed by DS_0. Given the size of hints $\mathcal{M}$, the algorithm allows DS_0 to preprocess $\mathbb{DB}$ and output the hints $\mathbb{HT}$ for subsequent online PIR queries.
- $\mathtt{HtPrep}(\mathbb{HT}) \to (\mathbb{K}, \mathbb{PH}_0, \mathbb{PH}_1)$: The algorithm is executed by a client. By inputting the received hints, the client runs the algorithm to create locally stored PIR keys $\mathbb{K}$ and generate the partial hints $\mathbb{PH}_{\beta\in\{0,1\}}$ which are outsourced to $DS_{\beta\in\{0,1\}}$, respectively.
- $\mathtt{Query}(\mathbb{K}, \theta) \to (\mathsf{qu}_0, \mathsf{qu}_1, \mathsf{st})$: The algorithm is executed by a client. Given the PIR keys $\mathbb{K}$ and a query index $\theta \in \{1, 2, \ldots, N\}$, the algorithm enables the client to formulate two PIR queries, denoted as qu_0 and qu_1, with a secret state st.
- $\mathtt{Answer}(\mathsf{qu}_\beta, \mathbb{PH}_\beta, \mathbb{DB}) \to \mathsf{ans}_\beta$: The algorithm is executed by $DS_{\beta\in\{0,1\}}$. With the inputs of the received PIR query qu_β, the outsourced hints $\mathbb{PH}_\beta$, and the database $\mathbb{DB}$, DS_β runs the algorithm to generate the corresponding PIR response ans_β.
- $\mathtt{Recov}(\mathsf{st}, \mathsf{ans}_0, \mathsf{ans}_1) \to (\mathsf{cnt}_\theta, \mathsf{aux})$: The algorithm is executed by a client. With the received PIR responses, ans_0 and ans_1, the client runs the algorithm to recover cnt_θ based on the secret state st, and obtain auxiliary information aux for hint refreshing.
- $\mathtt{HtUpd}(\mathsf{st}, \mathbb{K}, \mathsf{aux}) \to (\mathbb{K}, \mathsf{rq}_0, \mathsf{rq}_1)$: The algorithm is executed by a client. Taking as inputs the secret state st, the current PIR keys $\mathbb{K}$, and the corresponding auxiliary information aux, the client runs the algorithm to update $\mathbb{K}$ and create two hint refreshing requests, denoted as rq_0 and rq_1.
- $\mathtt{HtRfsh}(\mathsf{rq}_\beta, \mathbb{PH}_\beta) \to \mathbb{PH}_\beta$: The algorithm is executed by $DS_{\beta\in\{0,1\}}$. Given the hint refreshing request rq_β, DS_β runs the algorithm to refresh current outsourced hints $\mathbb{PH}_\beta$.

Correctness of two-server offline/online PIR: We say that a two-server offline/online PIR scheme is correct if, given a database, $\mathbb{DB}$, and an index $\theta \in \{1, 2, \ldots, n\}$, the probability

$$\Pr\left[\mathsf{cnt}'_\theta = \mathsf{cnt}_\theta : \begin{array}{l} \mathbb{HT} \leftarrow \mathtt{HtGen}(\mathcal{M}, \mathbb{DB}) \\ (\mathbb{K}, \mathbb{PH}_0, \mathbb{PH}_1) \leftarrow \mathtt{HtPrep}(\mathbb{HT}) \\ (\mathsf{qu}_0, \mathsf{qu}_1, \alpha) \leftarrow \mathtt{Query}(\mathbb{K}, \theta) \\ \mathsf{ans}_0 \leftarrow \mathtt{Answer}(\mathsf{qu}_0, \mathbb{PH}_0, \mathbb{DB}) \\ \mathsf{ans}_1 \leftarrow \mathtt{Answer}(\mathsf{qu}_1, \mathbb{PH}_1, \mathbb{DB}) \\ (\mathsf{cnt}'_\theta, \_) \leftarrow \mathtt{Recov}(\alpha, \mathsf{ans}_0, \mathsf{ans}_1) \end{array}\right]$$

is $1 - \eta$, where η is negligible.

Security Model: The security of the two-server offline/online PIR is captured by a family of stateful leakage functions $\mathcal{L} = (\mathcal{L}^{\mathrm{hprep}}, \mathcal{L}^{\mathrm{qu}}, \mathcal{L}^{\mathrm{upd}})$ that describes what information is leaked to a polynomial-probabilistic-time (PPT) adversary $\mathcal{A}$ during the executions of $\mathtt{HtPrep}(\cdot)$, $\mathtt{Query}(\cdot)$, and $\mathtt{HtUpd}(\cdot)$. The

security is formulated using a simulation that involves a real experiment $\mathsf{REAL}^{\Pi}_{\mathcal{A}}(\lambda)$ and an ideal experiment $\mathsf{IDEAL}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)$, where λ is a security parameter. If $\mathcal{A}$ cannot distinguish the two experiments (i.e., whether it interacts with a simulator $\mathcal{S}$ or a real client), there is no other information revealed except the information included in $\mathcal{L}$. The real experiment and ideal experiment are defined as follows:

- $\mathsf{REAL}^{\Pi}_{\mathcal{A}}(\lambda)$ - ① $\mathcal{A}$ chooses a database $\mathbb{DB}$, and if $\mathcal{A}$ controls DS_0, it executes $\mathtt{HtGen}(\cdot)$ and sends $\mathbb{HT}$ to a real client; ② the client executes $\mathtt{HtPrep}(\cdot)$ and outsources hints to $\mathcal{A}$; ③ $\mathcal{A}$ chooses a polynomial number of queries. For each chosen query, $\mathcal{S}$ executes $\mathtt{Query}(\cdot)$ and $\mathtt{HtUpd}(\cdot)$, and sends the corresponding PIR query and the hint refreshing request to $\mathcal{A}$; ④ Finally, $\mathcal{A}$ outputs a bit $\{0,1\}$.
- $\mathsf{IDEAL}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)$ - ① $\mathcal{A}$ chooses a database $\mathbb{DB}$, and if $\mathcal{A}$ controls DS_0, it sends $\mathcal{L}^{\text{hprep}}(\cdot)$ to $\mathcal{S}$; ② Using $\mathcal{L}^{\text{hprep}}(\cdot)$, $\mathcal{S}$ simulates as an honest client and generates the transcripts of executing $\mathtt{HtPrep}(\cdot)$; ③ $\mathcal{A}$ chooses a polynomial number of queries. For each chosen query, using $\mathcal{L}^{\text{qu}}(\cdot)$ and $\mathcal{L}^{\text{upd}}(\cdot)$, $\mathcal{S}$ simulates all transcripts during the execution of $\mathtt{Query}(\cdot)$ and $\mathtt{HtUpd}(\cdot)$; ④ Finally, $\mathcal{A}$ outputs a bit $\{0,1\}$.

Definition 1 ($\mathcal{L}$-Security): A two-server offline/online PIR scheme, Π, is $\mathcal{L}$-secure if, for a PPT adversary, $\mathcal{A}$, there exists a PPT algorithm $\mathcal{S}$ such that

$$|\Pr[\mathsf{REAL}^{\Pi}_{\mathcal{A}}(\lambda)=1]-\Pr[\mathsf{IDEAL}^{\Pi}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)=1]|\leq \mathtt{negl}(\lambda).$$

We argue that Π is adaptively secure if $\mathcal{A}$ has the capability to select its queries in an adaptive manner. In other words, $\mathcal{A}$ is not required to choose all queries at the beginning; instead, it can determine each query based on the transcripts received from previous queries.

D. Design Goals

In the context of the system and security models, we aim to propose a two-server PIR scheme that is both $\mathcal{L}$-adaptive secure and practical. Our focus on practicality is multi-faceted:

- *Large Database Support:* The proposed scheme should accommodate large-scale databases, with millions of records and KB-level data sizes.
- *Small Client Storage:* The proposed scheme should eliminate the need for a client to store a significant amount of data locally.
- *End-to-End Efficiency.* We define this metric as the per-query wall-clock latency, measured from issuing a query to retrieving the desired record. It accounts for client- and server-side computation while excluding network propagation delays. The scheme should achieve low end-to-end latency.
- *High Server Throughput:* The proposed scheme should enable database servers to efficiently process and respond to clients' PIR queries. We denote a server's throughput as the number of queries processed per second for each CPU core (#/s/core).
- *Communication Efficiency.* We define this metric as the per-query communication volume, i.e., the total number of bytes transmitted in both uplink and downlink. The scheme is expected to maintain communication efficiency with a low communication volume.

III. Preliminaries

A. Super-Increasing Sequence

A super-increasing sequence (SIS) is a set of integer numbers where each element is greater than the sum of its predecessors. Such a sequence can be denoted by $\mathbf{a}$ of size n. The SIS generation algorithm is defined as follows:

- $\mathtt{SIS.Gen}(n,\tau)\rightarrow \mathbf{a}$: Given a size n and a stride $\tau\in\mathbb{N}$, the algorithm outputs a super-increasing sequence $\mathbf{a}$, satisfying $\mathbf{a}[i]\cdot\tau<\mathbf{a}[i+1]$ for any $i\in[1,n-1]$.

B. Beaver's Matrix Multiplication

Beaver's multiplication technique is widely used in secure multi-party computation, which achieves secure multiplication of two secret-shared elements in a finite field $\mathbb{F}$. The technique has been generalized to secure multiplication of two secret-shared matrices in $\mathbb{F}$ [26]. In a two-party model (Party 0 and Party 1), given two additive secret shares of matrices $\mathbf{M}_0\in\mathbb{F}$ and $\mathbf{M}_1\in\mathbb{F}$, namely, $(\mathbf{M}^0_0,\mathbf{M}^1_0)$ and $(\mathbf{M}^0_1,\mathbf{M}^1_1)$ satisfying $\mathbf{M}^0_0+\mathbf{M}^1_0=\mathbf{M}_0$ and $\mathbf{M}^0_1+\mathbf{M}^1_1=\mathbf{M}_1$, Beaver's matrix multiplication can be captured by the following three algorithms:

- $\mathtt{BV.Gen}(\mathsf{dim}_0,\mathsf{dim}_1)\rightarrow(\mathbf{U}^0,\mathbf{U}^1,\mathbf{V}^0,\mathbf{V}^1,\mathbf{W}^0,\mathbf{W}^1)$: The algorithm is executed by a trusted dealer. It takes the dimensions of $\mathbf{M}_0$ and $\mathbf{M}_1$ as inputs, and outputs the additive secret shares of three random matrices $(\mathbf{U},\mathbf{V},\mathbf{W})$, satisfying $\mathbf{U}\cdot\mathbf{V}=\mathbf{W}$, where $\mathbf{U}$ and $\mathbf{V}$'s dimensions are dim_0 and dim_1, respectively.
- $\mathtt{BV.Prep}(\mathbf{M}^\beta_0,\mathbf{M}^\beta_1,\mathbf{U}^\beta,\mathbf{V}^\beta)\rightarrow(\mathbf{E}^\beta_0,\mathbf{E}^\beta_1)$: The algorithm is run by Party $\beta\in\{0,1\}$, with the inputs of secret-shared matrices, and outputs two preprocessed matrices $\mathbf{E}^\beta_0=\mathbf{M}^\beta_0-\mathbf{U}^\beta$ and $\mathbf{E}^\beta_1=\mathbf{M}^\beta_1-\mathbf{V}^\beta$ shared with Party $\bar{\beta}$.
- $\mathtt{BV.Proc}(\mathbf{U}^\beta,\mathbf{V}^\beta,\mathbf{E}^\beta_0,\mathbf{E}^\beta_1,\mathbf{E}^{\bar{\beta}}_0,\mathbf{E}^{\bar{\beta}}_1,\mathbf{W}^\beta)\rightarrow\mathbf{F}^\beta$: The algorithm is executed by Party $\beta\in\{0,1\}$, with the inputs of secret-shared matrices and the preprocessed matrices received from Party $\bar{\beta}$. In the algorithm, Party b reconstructs $\mathbf{E}^0=\mathbf{E}^\beta_0+\mathbf{E}^{\bar{\beta}}_0$ and $\mathbf{E}^1=\mathbf{E}^\beta_1+\mathbf{E}^{\bar{\beta}}_1$ and generates another matrix $\mathbf{F}_\beta=\beta\cdot(\mathbf{E}^0\cdot\mathbf{E}^1)+\mathbf{U}^\beta\cdot\mathbf{E}^1+\mathbf{E}^0\cdot\mathbf{V}^\beta+\mathbf{W}^\beta$.

Here, we have $\mathbf{F}_0+\mathbf{F}_1=\mathbf{M}_1\cdot\mathbf{M}_2\in\mathbb{F}$, and its security under the semi-honest and malicious models has been proved. We refer more details to [26].

Definition 2 (Simulation Security of Two-Party Beaver's Matrix Multiplication [26]): Suppose one party is compromised by a probabilistic polynomial time (PPT) adversary $\mathcal{A}$, two-party Beaver's matrix multiplication is secure if, given the leakage of the $(\mathcal{L}^{\text{BV}}(\mathbf{M}_0,\mathbf{M}_1)=(\mathsf{dim}_0,\mathsf{dim}_1)$, there exists a PPT algorithm, $\mathtt{SIM}$, that can make the outputs of the following two experiments computationally indistinguishable:

- $\mathsf{REAL}^{\text{BV}}(\lambda)$: This is the real-world experiment that outputs $(\mathbf{U}^\beta,\mathbf{V}^\beta,\mathbf{W}^\beta,\mathbf{E}^{\bar{\beta}}_0,\mathbf{E}^{\bar{\beta}}_1,\mathbf{F}^\beta)$ by running $\mathtt{BV.Gen}(\cdot)$, $\mathtt{BV.Prep}(\cdot)$, and $\mathtt{BV.Proc}(\cdot)$.
- $\mathsf{IDEAL}^{\text{BV}}(\lambda)$: This is the ideal experiment that outputs $\mathtt{SIM}(\mathcal{L}^{\text{BV}}(\mathbf{M}_0,\mathbf{M}_1))\rightarrow(\mathbf{U}^\beta,\mathbf{V}^\beta,\mathbf{W}^\beta,\mathbf{E}^{\bar{\beta}}_0,\mathbf{E}^{\bar{\beta}}_1,\mathbf{F}^\beta)$.

C. Bloom Filter

A Bloom Filter (BF) [27] is a compact data structure that utilizes an m-bit binary vector ($\mathbf{b}$) to denote a set of n variable-length elements $\mathbb{S}$. It enables efficient membership inquiries on an element s without false negatives and small false positives. There exist three algorithms for constructing a BF and achieving membership test:

- $\texttt{BF.Setup}(\epsilon, n, k) \rightarrow (\mathbb{H}, m)$: With the inputs of false positive rate $\epsilon \in (0,1)$, capacity n, and number of hash functions k, the algorithm outputs k universal hash functions $\mathbb{H}$ where each $h_j(\cdot) \rightarrow [1,m]$ for $j=1$ to k and the BF size m.
- $\texttt{BF.Build}(\mathbb{S}, \mathbb{H}, m) \rightarrow \mathbf{b}$: On the inputs of hash functions $\mathbb{H}$, the algorithm initializes an m-bit zero-binary, $\mathbf{b}$, traverses each $\mathbb{S}[i]$ for $i=1$ to m, sets $\mathbf{b}[h_{j\in[1,k]}(\mathbb{S}[i])]=1$ for $j=1$ to k, and outputs the m-bit binary $\mathbf{b}$.
- $\texttt{BF.Check}(s, \mathbf{b}) \rightarrow 0/1$: The algorithm takes an element s and the binary $\mathbf{b}$ as inputs. If $s \in \mathbb{S}$, i.e., $\mathbf{b}[h_j(s)]=1$ for $j=1$ to k, the algorithm outputs 1; otherwise, outputs 0.

False Positives of a BF: The false positive rate of a BF with the parameters (n, m, k) is:

$$\epsilon = \left(1 - \exp\left(-\frac{nk}{m}\right)\right)^k. \tag{1}$$

D. Pseudorandom Function

Pseudorandom function (PRF) is a cryptographic primitive used to generate outputs from inputs and one key, and can ensure that the generated outputs are computationally indistinguishable from truly random outputs. PRF is denoted by $\texttt{PRF}: \kappa \times \mathbb{D}_{\text{in}} \rightarrow \mathbb{D}_{\text{out}}$, where $\kappa \in \{0,1\}^\lambda$ is a symmetric key, and $\mathbb{D}_{\text{in}}$ and $\mathbb{D}_{\text{out}}$ are input and output domains.

E. Distributed Point Function

A two-party Distributed Point Function (DPF) [28] allows a client to divide *a point function*, denoted as $f(\cdot): x^* \rightarrow y^*$, where $x^* \in \mathbb{X}$ and $y^* \in \mathbb{Y}$, into two function shares: K_0 and K_1. These shares are then distributed to two parties, respectively, and reveal no information about the point function $f(\cdot)$. When combined, their evaluations at a specific point x^* can be used to recover $f(x^*)=y^*$, while evaluations at all other points $x' \neq x^*$ result in zeros. DPF can be captured by two algorithms:

- $\texttt{DPF.Gen}(\lambda, x^*, y^*) \rightarrow (K_0, K_1)$: The generation algorithm takes (x^*, y^*) and a security parameter λ as inputs, and outputs two function shares K_0 and K_1.
- $\texttt{DPF.Eval}(K_\beta, x) \rightarrow y_\beta$: On the inputs of one point $x \in \mathbf{x}$, and one function share K_β where $\beta \in \{0,1\}$, the evaluation algorithm outputs y_β.

Here, we have $\oplus_{\beta\in\{0,1\}} y_\beta = y_0 \oplus y_1 = f(x)$. If $x \neq x^*$, $f(x)=0$; otherwise $f(x)=y^*$.

Definition 3 (Simulation Security of Two-Party DPF [28]): Suppose one party is compromised by a PPT adversary $\mathcal{A}$, two-party DPF is secure if, given the leakage of the input and output sizes $(\mathcal{L}^{\text{DPF}}(f) = (|\mathbb{X}|, |\mathbb{Y}|)$, there exists a PPT algorithm, $\texttt{SIM}$, that can make the outputs of the following two experiments computationally indistinguishable:

- $\text{REAL}^{\text{DPF}}(\lambda)$: This is the real-world experiment that outputs (K_0, K_1) by running $\texttt{DPF.Gen}(\cdot)$.
- $\text{IDEAL}^{\text{DPF}}(\lambda)$: This is the ideal experiment that outputs $\texttt{SIM}(\mathcal{L}^{\text{DPF}}(f)) \rightarrow (K_0, K_1)$.

F. Puncturable Pseudorandom Sets

Puncturable pseudorandom set (PPRS) is a cryptographic primitive used to generate a compressed representation of a pseudorandom set $\mathbb{S} \subseteq \{1,2,\ldots,n\}$ and $|\mathbb{S}| = \sqrt{n}$, which consists of five algorithms:

- $\texttt{PPRS.Gen}(\lambda, n) \rightarrow \mathsf{sk}$: The algorithm generates a uniformly random puncturable-set key $\mathsf{sk} \in \{0,1\}^\lambda$ by taking a security parameter λ and the size of inputs n.
- $\texttt{PPRS.Eval}_{\text{I}}(\mathsf{sk} \backslash \mathsf{sk}^*) \rightarrow \mathbb{S}$: On the inputs of sk or sk^*, the algorithm outputs a pseudorandom set $\mathbb{S} \subseteq \{1,2,\ldots,n\}$.
- $\texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk} \backslash \mathsf{sk}^*, \zeta) \rightarrow \mathbb{S}$: On the inputs of sk or sk^* and an offset $\zeta \in [1,n]$, the algorithm outputs a pseudorandom set $\mathbb{S} \subseteq \{1,2,\ldots,n\}$, where $\mathbb{S} = \{(i+\zeta) \bmod n | i \in \texttt{PPRS.Eval}_{\text{I}}(\mathsf{sk} \backslash \mathsf{sk}^*)\}$.
- $\texttt{PPRS.Punc}(\mathsf{sk}, \zeta, i^*) \rightarrow \mathsf{sk}^*$: On the inputs of sk and i^*, the algorithm outputs a punctured set key sk^*, satisfying $\texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}^*, \zeta) = \texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}, \zeta) \backslash \{i^*\}$.
- $\texttt{PPRS.Loc}(\mathsf{sk}, i) \rightarrow \zeta$: On the inputs of sk and an element $i \in \{1,2,\ldots,n\}$, the algorithm randomly picks $r \in [1, \sqrt{n}]$, and outputs an offset ζ such that $i = \texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}, \zeta)[r]$.

The security of PPRS can be captured by a game between a PPT adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

$\mathbf{Expt}^{\text{Ind}}_{\text{PPRS},\mathcal{A}}$: Security Game for PPRS
- $\mathcal{C}$ executes the following steps:
  - $\mathsf{sk} \leftarrow \texttt{PPRS.Gen}(\lambda, n)$, $\mathbb{S} \leftarrow \texttt{PPRS.Eval}_{\text{I}}(\mathsf{sk})$,
  - randomly pick $i^* \in \mathbb{S}$, $\mathsf{sk}^* \leftarrow \texttt{PPRS.Punc}(\mathsf{sk}, 0, i^*)$

  and sends (n, sk^*) to $\mathcal{A}$.
- $\mathcal{A}$ selects $i' \in \{1,2,\ldots,n\}$.
- $\mathcal{A}$ outputs 1 if $i'=i^*$; otherwise 0.

Definition 4 (Security of PPRS [12]): PPRS is computationally secure if for a PPT adversary $\mathcal{A}$, we have $Pr[\mathbf{Expt}^{\text{Ind}}_{\text{PPRS},\mathcal{A}}(\lambda)=1] - \frac{1}{n-|\mathbb{S}|+1} = \texttt{negl}(\lambda)$, where λ is the security parameter.

$\mathbf{Expt}^{\text{Prand}}_{\text{PPRS},\mathcal{A}}$: Security Game for PPRS
- $\mathcal{C}$ executes the following steps:
  - $\gamma \xleftarrow{R} \{0,1\}$
  - $\mathsf{sk} \leftarrow \texttt{PPRS.Gen}(\lambda, n)$, $\mathbb{S}_\gamma \leftarrow \texttt{PPRS.Eval}_{\text{I}}(\mathsf{sk})$
  - $\mathbb{S}_{\bar{\gamma}} \xleftarrow{R} \binom{n}{\sqrt{n}}$

  and sends $(n, \mathbb{S}_0, \mathbb{S}_1)$ to $\mathcal{A}$.
- $\mathcal{A}$ picks $\beta \in \{0,1\}$.
- $\mathcal{A}$ outputs 1 if $b=\gamma$; otherwise 0.

Definition 5 (Pseudorandomness of PPRS [12]): PPRS implies pseudorandomness if for a PPT adversary $\mathcal{A}$, we have $Pr[\mathbf{Expt}^{\text{Prand}}_{\text{PPRS},\mathcal{A}}(\lambda)=1] - \frac{1}{2} = \texttt{negl}(\lambda)$, where λ is the security parameter.

The above two definitions also hold if $\mathbb{S}$ (resp. $\mathbb{S}_\gamma$) is sampled based on (sk, ζ), i.e., $\mathbb{S} = \mathtt{PPRS.Eval}_{\mathrm{II}}(\mathsf{sk}, \zeta)$, where $\mathsf{sk} \leftarrow \mathtt{PPRS.Gen}(\lambda)$ and $\zeta \xleftarrow{R} [1, n]$.

G. LWE-Based Linearly Homomorphic Encryption

A linearly homomorphic encryption scheme with preprocessing (LHEP) [9] enables efficient and secure computation of the dot product between a ciphertext vector/matrix $\mathbf{c}$, and a plaintext vector/matrix $\mathbf{b}$, and its security is based on Learning-With-Errors (LWE) assumptions. LHEP consists of the following five algorithms:

- $\mathtt{LHEP.Setup}(\lambda) \rightarrow \mathsf{pp}$: Given a security parameter λ, the algorithm outputs public parameters $\mathsf{pp} = \{n, q, p, \chi, m, \Delta, \mathbf{A}\}$, where (n, q, χ) are LWE parameters, p is the plaintext modulus, m is the plaintext vector's dimension, $\Delta = \lfloor q/p \rfloor$, and $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ is a random LWE matrix.
- $\mathtt{LHEP.Enc}(\mathsf{pp}, \mathbf{t}, \mathbf{v}) \rightarrow \mathbf{c}$: On the inputs of a plaintext vector $\mathbf{v} \in \mathbb{Z}_p^m$ and a secret key $\mathbf{t} \in \mathbb{Z}_q^n$, the algorithm samples $\mathbf{e} \in \chi^m$, and outputs $\mathbf{c} = \mathbf{A} \cdot \mathbf{t} + \mathbf{e} + \Delta \cdot \mathbf{v} \bmod q$.
- $\mathtt{LHEP.Prep}(\mathsf{pp}, \mathbf{b}) \rightarrow \mathbf{p}$: With the inputs of a plaintext vector $\mathbf{b} \in \mathbb{Z}_p^{1 \times m}$, the algorithm outputs $\mathbf{p} = \mathbf{b} \cdot \mathbf{A} \bmod q$.
- $\mathtt{LHEP.Apply}(\mathsf{pp}, \mathbf{b}, \mathbf{c}) \rightarrow \mathsf{ct}$: On the inputs of $\mathbf{b}$ and $\mathbf{c}$, the algorithm outputs the ciphertext of $< \mathbf{v}, \mathbf{b} >$, $\mathsf{ct} = \mathbf{b} \cdot \mathbf{c} \bmod q$.
- $\mathtt{LHEP.Dec}(\mathsf{pp}, \mathsf{ct}, \mathbf{p}, \mathbf{t}) \rightarrow d$: With the inputs of ct, $\mathbf{p}$, and $\mathbf{t}$, the algorithm outputs $d = \frac{\mathtt{Round}_\Delta((\mathsf{ct} - \mathbf{p} \cdot \mathbf{t}) \bmod q)}{\Delta} \bmod p$, where $\mathtt{Round}_\Delta(\cdot)$ is a function that rounds an input to the nearest integral multiple of Δ.

LWE Parameters: The LWE parameters (n, q, χ) are defined as follows: n and q are positive integers, and χ denotes a discrete Gaussian distribution with standard deviation σ and mean 0. Given the concrete values of LWE parameters, the hardness of solving a decision/search LWE problem can be estimated according to [29].

Definition 6 (Security of LHEP [9]): LHEP is semantically secure if, given public parameters $\mathsf{pp} \leftarrow \mathtt{Setup}(\lambda)$ and two vectors $\mathbf{v}_0, \mathbf{v}_1 \in \mathbb{Z}_p^m$, for a PPT adversary $\mathcal{A}$, we have $|W_0 - W_1| = \mathtt{negl}(\lambda)$, where

$$W_{\beta \in \{0,1\}} = Pr\left[[\mathcal{A}(\mathsf{pp}, \mathbf{c}) = 1 : \begin{array}{c} \mathsf{sk} \leftarrow \mathbb{Z}_q^n \\ \mathbf{c} \leftarrow \mathtt{Enc}(\mathsf{pp}, \mathsf{sk}, \mathbf{v}_\beta) \end{array} \right].$$

IV. Proposed Scheme: PIRS

In this section, we present the construction of the proposed two-server offline/online private information scheme, named PIRS.

A. Scheme Design: Outsourcing Hints

To improve readability, Fig. 2 provides a high-level overview of PIRS. The details of PIRS can be found in Scheme 1, which allows a client to retrieve records from a remote database without exposing query indices, including seven algorithms. In the following, we provide a design sketch and highlight our scheme's novelty.

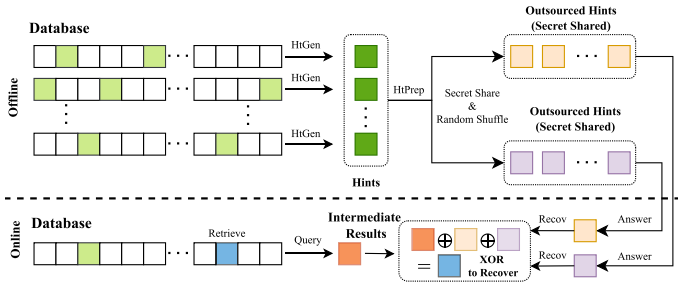


Fig. 2. High-level overview of PIRS. The offline phase includes hint and key generation, parity word construction, and outsourcing to the two servers. The online phase includes PIR queries and record reconstruction. The main algorithms and data flows are highlighted using arrows.

PIRS preprocesses the entire database during an offline phase to improve retrieval efficiency during an online phase. Compared to existing two server offline/online PIR schemes [12], [13], it distinguishes itself by eliminating the requirement of storing numerous hints on the client's end through outsourcing hints to the servers. Impressively, PIRS accomplishes this without compromising the sublinear online computational complexity on the server side. As such, PIRS is an extension and hybrid of several techniques, including punctuable pseudorandom set (PPRS), secret sharing, randomization, distributed point functon (DPF), and oblivious switching.

Specifically, during the offline phase, the server DS_0 initializes the scheme by generating a set of puncturable-set keys $(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_\mathcal{M})$ with $\mathtt{PPRS.Gen}(\cdot)$. For each key $\mathsf{sk}_{i \in [1,\mathcal{M}]}$, there exists an associated random offset $\zeta_i \in [1, N]$. Combining sk_i and ζ_i as a PIR key, a pseudorandom set $\mathbb{S}_i = \mathtt{PPRS.Eval}_{\mathrm{II}}(\mathsf{sk}_i, \zeta_i)$ can be found to create a Parity Word of $\mathbb{DB}$, denoted as $\mathsf{PW}_i = \oplus_{j \in \mathbb{S}_i} \mathsf{cnt}_j$. In existing two-server offline/online PIR schemes [12], [13], a client has to keep all the hints $\mathbb{HT} = (\mathsf{sk}_i, \zeta_i, \mathsf{PW}_i)_{i=1}^\mathcal{M}$ locally. The storage complexity is $\mathcal{O}(\mathcal{M} \cdot (\lambda + \log_2(N) + \ell))$, directly proportional to the voluminous data sizes encountered in the entire database $\mathbb{DB}$, which could be incredibly large. In comparison, PIRS is an alternative with more focus on lightweight client storage: it outsources partial hints (the Parity Words $(\mathsf{PW}_i)_{i=1}^\mathcal{M}$) to the servers, allowing clients to merely store the PIR keys $\mathbb{K} = (\mathsf{vsk}, (\mathsf{sk}_i, \zeta_i, \mathsf{ver}_i)_{i=1}^\mathcal{M})$. Here, vsk is a secret key for hint updating, which will be discussed later. In this way, the storage overheads on the client's end can be significantly reduced to $\mathcal{O}(\lambda + \mathcal{M} \cdot (\lambda + \log_2 N + |\mathsf{ver}|))$.

To securely outsource partial hints, PIRS applies secret sharing and randomization techniques: All Parity Words $(\mathsf{PW}_i)_{i=1}^\mathcal{M}$ are randomly shuffled by a permutation function $\phi(\cdot)$, and are randomly divided into two secret shares $\mathbb{PH}_0 = (\mathsf{PW}_i^0)_{i=1}^\mathcal{M}$ and $\mathbb{PH}_1 = (\mathsf{PW}_i^1)_{i=1}^\mathcal{M}$, satisfying $\mathsf{PW}_i^0 \oplus \mathsf{PW}_i^1 = \mathsf{PW}_i$. By doing so, the relationship between the original hints $\mathbb{HT}$ and the outsourced partial hints $\mathbb{PH}_0$ and $\mathbb{PH}_1$ can be concealed. The design implicitly ensures that the client can recover a Parity Word $\mathsf{PW}_{i \in [1,\mathcal{M}]}$ with the assistance of the servers without

Scheme 1: PIRS: Two-Server Private Information Retrieval.

Algorithm $\texttt{HtGen}(\mathcal{M}, \mathbb{DB})$
  Parse $\mathbb{DB}$ as $(\mathsf{cnt}_i)_{i=1}^{N}$
  for $i = 1$ to $\mathcal{M}$ **do**
    $\mathsf{sk}_i \leftarrow \texttt{PPRS.Gen}(\lambda, N)$, $\zeta_i \xleftarrow{\mathcal{R}} [1, N]$
    $\mathbb{S}_i \leftarrow \texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}_i, \zeta_i)$
    $\mathsf{PW}_i \leftarrow \oplus_{j \in \mathbb{S}_i} \mathsf{cnt}_j$
  return $\mathbb{HT} \leftarrow (\mathsf{sk}_i, \zeta_i, \mathsf{PW}_i)_{i=1}^{\mathcal{M}}$
Algorithm $\texttt{HtPrep}(\mathbb{HT})$
  Parse $\mathbb{HT}$ as $(\mathsf{sk}_i, \zeta_i, \mathsf{PW}_i)_{i=1}^{\mathcal{M}}$
  Choose a random permutation $\phi(\cdot) : [1, \mathcal{M}] \to [1, \mathcal{M}]$
  Permute $(\mathsf{sk}_i, \zeta_i, \mathsf{PW}_i)_{i=1}^{\mathcal{M}}$ using $\phi(\cdot)$
  $\mathsf{vsk} \leftarrow \{0,1\}^{\lambda}$
  for $i = 1$ to $\mathcal{M}$ **do**
    $\mathsf{ver}_i \leftarrow 0$, $\mathsf{PW}_i^0 \xleftarrow{\mathcal{R}} \{0,1\}^{\ell}$, $\mathsf{PW}_i^1 \leftarrow \mathsf{PW}_i \oplus \mathsf{PW}_i^0$
  $\mathbb{PH}_0 \leftarrow (\mathsf{PW}_i^0)_{i=1}^{\mathcal{M}}$, $\mathbb{PH}_1 \leftarrow (\mathsf{PW}_i^0)_{i=1}^{\mathcal{M}}$
  $\mathbb{K} \leftarrow (\mathsf{vsk}, (\mathsf{sk}_i, \zeta_i, \mathsf{ver}_i)_{i=1}^{\mathcal{M}})$
  return $(\mathbb{K}, \mathbb{PH})$
Algorithm $\texttt{Query}(\mathbb{K}, \theta)$
  Parse $\mathbb{K}$ as $(\mathsf{vsk}, (\mathsf{sk}_i, \zeta_i, \mathsf{ver}_i)_{i=1}^{\mathcal{M}})$
  Find $\delta \in [1, \mathcal{M}]$ satisfying $\theta \in \texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}_\delta, \zeta_\delta)$
  Sample $\alpha \leftarrow \texttt{Bernoulli}(\frac{2(\sqrt{N}-1)}{N})$
  $\mathsf{sk}_{\text{new}} \leftarrow \texttt{PPRS.Gen}(\lambda, N)$,
  $\zeta_{\text{new}} \leftarrow \texttt{PPRS.Loc}(\mathsf{sk}_{\text{new}}, \theta)$
  $\mathbb{S}_{\text{new}} \leftarrow \texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}_{\text{new}}, \zeta_{\text{new}})$
  $\mathsf{sk}_0' \leftarrow \texttt{PPRS.Gen}(\lambda, N)$, $\zeta_0' \xleftarrow{\mathcal{R}} [1, N]$
  $\mathsf{sk}_1' \leftarrow \texttt{PPRS.Gen}(\lambda, N)$, $\zeta_1' \xleftarrow{\mathcal{R}} [1, N]$
  if $\alpha = 0$ **then**
    $\rho \xleftarrow{\mathcal{R}} [1, \mathcal{M}]$, $\varkappa_1 \leftarrow \varkappa_0 \leftarrow \theta$
    $\mathbb{S}_0 \leftarrow \mathbb{S}_{\text{new}}$, $r_0 \xleftarrow{\mathcal{R}} \mathbb{S}_0 \backslash \{\varkappa_0\}$
    $\mathbb{S}_1 \leftarrow \texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}_\delta, \zeta_\delta)$, $r_1 \xleftarrow{\mathcal{R}} \mathbb{S}_1 \backslash \{\varkappa_1\}$
    $\mathsf{sk}_0^* \leftarrow \texttt{PPRS.Punc}(\mathsf{sk}_{\text{new}}, \zeta_{\text{new}}, \varkappa_0)$, $\zeta_0^* \leftarrow \zeta_{\text{new}}$
    $\mathsf{sk}_1^* \leftarrow \texttt{PPRS.Punc}(\mathsf{sk}_\delta, \zeta_\delta, \varkappa_1)$, $\zeta_1^* \leftarrow \zeta_\delta$
    if $\rho \neq \delta$ **then**
      $\mathsf{sk}_1' \leftarrow \mathsf{sk}_\rho$, $\zeta_1' \leftarrow \zeta_\rho$
  else if $\alpha = 1$ **then**
    $\mathbb{S}_1 \leftarrow \mathbb{S}_0 \leftarrow \mathbb{S}_{\text{new}}$, $\zeta_1^* \leftarrow \zeta_0^* \leftarrow \zeta_{\text{new}}$
    $\gamma \xleftarrow{\mathcal{R}} \{0,1\}$, $\delta \xleftarrow{\mathcal{R}} [1, \mathcal{M}]$
    $\varkappa_\gamma \leftarrow \theta$, $r_\gamma \xleftarrow{\mathcal{R}} \mathbb{S}_\gamma \backslash \{\varkappa_\gamma\}$, $\varkappa_{\bar{\gamma}} \leftarrow r_\gamma$, $r_{\bar{\gamma}} \xleftarrow{\mathcal{R}} \mathbb{S}_{\bar{\gamma}} \backslash \{\varkappa_{\bar{\gamma}}\}$
    $\mathsf{sk}_\gamma^* \leftarrow \texttt{PPRS.Punc}(\mathsf{sk}_{\text{new}}, \zeta_{\text{new}}, \varkappa_\gamma)$
    $\mathsf{sk}_{\bar{\gamma}}^* \leftarrow \texttt{PPRS.Punc}(\mathsf{sk}_{\text{new}}, \zeta_{\text{new}}, \varkappa_{\bar{\gamma}})$
  $\mathsf{qu}_0 \leftarrow (\delta, \mathsf{sk}_0', \zeta_0', \mathsf{sk}_0^*, \zeta_0^*, r_0)$,
  $\mathsf{qu}_1 \leftarrow (\delta, \mathsf{sk}_1', \zeta_1', \mathsf{sk}_1^*, \zeta_1^*, r_1)$
  $\mathsf{st} \leftarrow (\alpha, \delta, \rho, \hat{sk}_0, \zeta_0', \mathsf{sk}_{\text{new}}, \zeta_{\text{new}})$
  return $(\mathsf{st}, \mathsf{qu}_0, \mathsf{qu}_1)$
Algorithm $\texttt{Answer}(\mathsf{qu}_\beta, \mathbb{PH}_\beta, \mathbb{DB})$
  Parse qu_β as $(\delta, \mathsf{sk}_\beta', \zeta_\beta', \mathsf{sk}_\beta^*, \zeta_\beta^*, r_\beta)$
  Parse $\mathbb{PH}_\beta$ and $\mathbb{DB}$ as $(\mathsf{PW}_i^\beta)_{i=1}^{\mathcal{M}}$ and $(\mathsf{cnt}_i)_{i=1}^{N}$
  $\mathbb{S}_\beta' \leftarrow \texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}_\beta', \zeta_\beta')$, $D_\beta' \leftarrow \oplus_{j \in \mathbb{S}_\beta'} \mathsf{cnt}_j$
  $\mathbb{S}_\beta \leftarrow \texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}_\beta^*, \zeta_\beta^*)$, $D_\beta \leftarrow \oplus_{j \in \mathbb{S}_\beta} \mathsf{cnt}_j$
  return $\mathsf{ans}_\beta \leftarrow (D_\beta', D_\beta, \mathsf{cnt}_{r_\beta}, \mathsf{PW}_\delta^\beta)$
Algorithm $\texttt{Recov}(\mathsf{st}, \mathsf{ans}_0, \mathsf{ans}_1)$
  Parse st as $(\alpha, \delta, \rho, \hat{sk}_0, \zeta_0', \mathsf{sk}_{\text{new}}, \zeta_{\text{new}})$
  $\mathsf{PW}_\delta \leftarrow \mathsf{PW}_\delta^0 \oplus \mathsf{PW}_\delta^1$, $y^* \leftarrow 0$, $\mathsf{PW}_{\text{new}} \xleftarrow{\mathcal{R}} \{0,1\}^{\ell}$
  if $\mathsf{ver}_\delta \neq 0$ **then**
    $\mathsf{PW}_\delta \leftarrow \mathsf{PW}_\delta \oplus \texttt{PRF}(\mathsf{vsk}, \delta || \mathsf{ver}_\delta)$
  if $\alpha = 0$ **then**
    $\mathsf{cnt}_\theta \leftarrow \mathsf{PW}_\delta \oplus D_1$
    $\mathsf{PW}_{\text{upd}} \leftarrow D_0 \oplus \mathsf{cnt}_\theta$
    if $\rho \neq \delta$ **then**
      $\mathsf{PW}_{\text{upd}} \leftarrow D_0'$
      if $\mathsf{ver}_\rho \neq 0$ **then**
        $\mathsf{PW}_{\text{new}} \leftarrow \texttt{PRF}(\mathsf{vsk}, \rho || \mathsf{ver}_\rho)$
      $\mathsf{ver}_\rho \leftarrow \mathsf{ver}_\rho + 1$
      $\mathsf{PW}_{\text{new}} \leftarrow$
      $\mathsf{PW}_{\text{new}} \oplus \mathsf{cnt}_\theta \oplus D_0 \oplus D_1' \oplus \texttt{PRF}(\mathsf{vsk}, \rho || \mathsf{ver}_\rho)$
  else if $\alpha = 1$ **then**
    $\mathsf{cnt}_\theta \leftarrow D_0 \oplus D_1 \oplus \mathsf{cnt}_{r_\gamma}$
    $\mathsf{PW}_{\text{upd}} \leftarrow \mathsf{PW}_\delta$
  $\mathsf{aux} \leftarrow (\mathsf{PW}_{\text{upd}}, \mathsf{PW}_{\text{new}}, y^*)$
  return $(\mathsf{cnt}_\theta, \mathsf{aux})$
Algorithm $\texttt{HtUpd}(\mathsf{st}, \mathbb{K}, \mathsf{aux})$
  Parse $\mathbb{K}$ and aux as $(\mathsf{sk}_i, \zeta_i, \mathsf{ver}_i)_{i=1}^{\mathcal{M}}$ and
  $(\mathsf{PW}_{\text{upd}}, \mathsf{PW}_{\text{new}}, y^*)$
  Parse st as $(\alpha, \delta, \rho, \mathsf{sk}_0', \zeta_0', \mathsf{sk}_{\text{new}}, \zeta_{\text{new}})$
  if $\alpha = 0$ **then**
    Update $\mathsf{sk}_\delta \leftarrow \mathsf{sk}_{\text{new}}$ and $\zeta_\delta \leftarrow \zeta_{\text{new}}$ in $\mathbb{K}$
    if $\rho \neq \delta$ **then**
      Update $\mathsf{sk}_\rho \leftarrow \mathsf{sk}_{\text{new}}$ and $\zeta_\rho \leftarrow \zeta_{\text{new}}$ in $\mathbb{K}$
      Update $\mathsf{sk}_\delta \leftarrow \mathsf{sk}_0'$ and $\zeta_\delta \leftarrow \zeta_0'$ in $\mathbb{K}$
  $\mathsf{PW}_{\text{upd}}^0 \xleftarrow{\mathcal{R}} \{0,1\}^{\ell}$, $\mathsf{PW}_{\text{upd}}^1 \leftarrow \mathsf{PW}_{\text{upd}}^0 \oplus \mathsf{PW}_{\text{upd}}$
  $(K_0, K_1) \leftarrow \texttt{DPF.Gen}(\lambda, \rho, y^*)$
  $\mathsf{rq}_0 \leftarrow (\delta, K_0, \mathsf{PW}_{\text{upd}}^0, \mathsf{PW}_{\text{new}})$,
  $\mathsf{rq}_1 \leftarrow (\delta, K_1, \mathsf{PW}_{\text{upd}}^1, \mathsf{PW}_{\text{new}})$
  return $(\mathbb{K}, \mathsf{rq}_0, \mathsf{rq}_1)$
Algorithm $\texttt{HtRfsh}(\mathsf{rq}_\beta, \mathbb{PH}_\beta)$
  Parse rq_β and $\mathbb{PH}_\beta$ as $(\delta, K_\beta, \mathsf{PW}_{\text{upd}}^\beta, \mathsf{PW}_{\text{new}})$ and
  $(\mathsf{PW}_i)_{i=1}^{\mathcal{M}}$
  Update $\mathsf{PW}_\delta^\beta \leftarrow \mathsf{PW}_{\text{upd}}^\beta$ in $\mathbb{PH}_\beta$
  for $i = 1$ to $\mathcal{M}$ **do**
    if $\texttt{DPF.Eval}(K_\beta, i) = 1$ **then**
      $\mathsf{PW}_i^\beta \leftarrow \mathsf{PW}_i^\beta \oplus \mathsf{PW}_{\text{new}}$
  return $\mathbb{PH}_\beta$

revealing its pre-shuffled position $\phi^{-1}(i)$. The feature is crucial, especially considering that DS_0 creates all Parity Words, yet the client needs to retrieve particular Parity Words associated with PIR queries.

After outsourcing the hint, the client is capable of retrieving the θ-th record cnt_θ from the servers privately where $\theta \in [1, N]$. The process begins with the client pinpointing the relevant PIR key (hint) $(\mathsf{sk}_\delta, \zeta_\delta)$ that fulfills the condition $\theta \in \texttt{PPRS.Eval}_{\text{II}}(\mathsf{sk}_\delta, \zeta_\delta)$, with $\delta \in [1, \mathcal{M}]$. Then, the client proceeds to construct two distinct PIR queries qu_0 and qu_1 for DS_0 and DS_1, respectively. Similar to the method in [13], the formulation of the PIR queries adheres to two

cases for the purpose of preventing potential information leakage:

- *Case-I ($\alpha = 0$):* The PIR query qu_0 incorporates the index δ, and the PIR query qu_1 embeds both the index δ and the set difference $\mathbb{S}_1\backslash\{\theta\}$, where $\mathbb{S}_1 = \mathtt{PPRS.Eval}_{\mathrm{II}}(\mathsf{sk}_\delta, \zeta_\delta)$. In response, the server DS_0 provides the partial hint PW_δ^0, and the server DS_1 returns the partial hint PW_δ^1 and the Parity Word D_1 of $\mathbb{DB}$ indexed by the set $\mathbb{S}_1\backslash\{\theta\}$. The client recovers the θ-th record as:

$$\begin{aligned}\mathsf{cnt}_\theta &= \mathsf{PW}_\delta^0 \oplus \mathsf{PW}_\delta^1 \oplus D_1 \\ &= (\oplus_{j\in\mathbb{S}_1}\mathsf{cnt}_j) \oplus (\oplus_{j\in\mathbb{S}_1\backslash\{\theta\}}\mathsf{cnt}_j). \end{aligned} \tag{2}$$

- *Case-II ($\alpha = 1$):* The PIR query qu_0 incorporates the set difference $\mathbb{S}_0\backslash\{\varkappa_0\}$ and a random index $r_0 \in \mathbb{S}_0\backslash\{\varkappa_0\}$, and the PIR query qu_1 incorporates a set $\mathbb{S}_1\backslash\{\varkappa_1\}$ and a random index $r_1 \in \mathbb{S}_1\backslash\{\varkappa_1\}$. The sets $\mathbb{S}_0$ and $\mathbb{S}_1$ are congruent, each comprising $\sqrt{N}$ unique indices from the sequence $\{1, 2, 3, \ldots, N\}$, one of which is the desired θ. With the configuration, $\varkappa_{\gamma\in\{0,1\}}$ corresponds to θ, and $\varkappa_{\bar{\gamma}} = r_\gamma$. Accordingly, the server DS_0 returns the Parity Word D_0 of $\mathbb{DB}$ indexed by the set difference $\mathbb{S}_0\backslash\{\varkappa_0\}$ and the r_0-th record cnt_{r_0}, and the server DS_1 returns the Parity Word D_1 of $\mathbb{DB}$ indexed by the set $\mathbb{S}_1\backslash\{\varkappa_1\}$ and the r_1-th record cnt_{r_1}. The client recovers the θ-th record as:

$$\begin{aligned}\mathsf{cnt}_\theta &= D_0 \oplus D_1 \oplus \mathsf{cnt}_{r_\gamma} \\ &= (\oplus_{j\in\mathbb{S}_1\backslash\{\varkappa_0\}}\mathsf{cnt}_j) \oplus (\oplus_{j\in\mathbb{S}_1\backslash\{\varkappa_1\}}\mathsf{cnt}_j) \oplus \mathsf{cnt}_{r_\gamma} \\ &= \mathsf{cnt}_\theta \oplus \mathsf{cnt}_{r_\gamma} \oplus \mathsf{cnt}_{r_\gamma}. \end{aligned} \tag{3}$$

To ensure security, it is crucial that the two cases above are indistinguishable to the servers, which necessitates the inclusion of dummy data within the PIR queries qu_0 and qu_1: 1) a random index $\delta \in [1, \mathcal{M}]$ should be incorporated into the PIR queries in Case-II; and 2) a set difference $\mathbb{S}_0\backslash\{\theta\}$ and corresponding random indices r_0 and r_1 should be incorporated into the PIR queries in Case-I. Moreover, the set difference in the PIR queries can be compressed using PPRS to save communication overheads, e.g., $\mathbb{S}_1\backslash\{\theta\}$ can be represented by $(\mathsf{sk}_\delta^*, \zeta_\delta^*)$.

After receiving PIR responses, the next step for the client is to update the previously used hints. However, refreshing outsourced hints is challenging. The disclosure of δ allows differentiating between two queries on the same record with non-negligible probability, if the client directly replaces PW_δ. To address the issue, we propose a technique named oblivious switching, which enables the client to obliviously switch PW_δ with another random Parity Word PW_ρ, where $\rho \in [1, \mathcal{M}]$. In other words, after hint refreshing, PW_ρ is fetched if the client queries θ again, instead of PW_δ. Although the technique is named oblivious switching, it is more than mere "switching" and "replacement", since the used hint PW_δ cannot be reused. The client must resample a new Parity Word to substitute the original PW_δ, and then swap the new Parity Word with PW_ρ without exposing ρ. Most importantly, the original distribution of the outsourced hints should not be altered during the process. To achieve this, we apply DPF and make subtle modifications to the PIR queries, which enables the client to obliviously rewrite PW_ρ. To further improve the efficiency of DPF, the secret key vsk and one-time pad are used to protect $\mathsf{PW}_{\mathrm{new}}$. By doing so, the result of each DPF evaluation is either 0 or 1 instead of ℓ. The design can avoid the time-consuming "expanding" operations for each DPF evaluation, when the data record size is large, i.e., ℓ is large. After refreshing, we have $\mathsf{ver}_\delta > 0$, and the client needs to compute a one-time pad $\mathsf{PRF}(\mathsf{vsk}, \rho||\mathsf{ver}_\delta)$ to recover PW_δ. Due to DPF's security, the relationship between the randomly picked ρ and the queried δ can be concealed during the refreshing. As a result, two PIR queries on the same record cannot be distinguished.

Lazy Hint Refreshing: PIRS leverages oblivious switching for outsourced hint refreshing, which entails a computational complexity of $\mathcal{O}(\mathcal{M})$ (xor $\mathcal{M}$ Parity Words to hide ρ) on the server side. Especially for large stored contents, the technique is more computationally demanding than existing two-server offline/online schemes [12], [13] and results in delays. Fortunately, the client can consecutively launch multiple PIR queries without initiating an immediate hint refresh (see correctness analysis in Section V-A). Instead, the accumulated used hints can be refreshed in a lazy manner. Each pair of PIR queries occupy two used hints with the indices δ and ρ, and these used hints should be removed for subsequent PIR queries until the hint refreshing is completed.

Supporting Mutable Databases: PIRS can incorporate the database update mechanism proposed in [13]. The mechanism enables a client to store the database as an array of sub-databases, with each sub-database having corresponding hints updated independently. The approach significantly reduces the hint updating costs in comparison to regenerating hints for the entire database. As PIR over mutable databases is not our paper's focus, we are not going to delve into it. Further details are available in [13].

B. Secure and Efficient Hint Locating

Although PIRS does not require a client to store numerous hints locally, the client needs to perform one indispensable operation: PIR keys $(\mathsf{sk}_i, \zeta_i)_{i=1}^{\mathcal{M}}$ must be expanded sequentially until a key (hint) $(\mathsf{sk}_\delta, \zeta_\delta)$ is located containing the query index θ (line 17in Scheme 1). The method is named *LinearScan*, and the expected time to find the correct hint is linear in $\mathcal{M}$ on the client side and $\sqrt{N}$ PRF operations are necessary for expanding each PIR key, which becomes an efficiency bottleneck. To reduce hint locating delay when generating PIR queries, a client can allocate $\log_2 N \cdot \mathcal{M} \cdot \sqrt{N}$ extra storage space to accommodate a "hashtable" (a reverse-lookup table) that preserves the relationship between indices and hints [12], [13], and the client can search the table to identify an appropriate key in a fast way. The method is named *HashStore*. To circumvent both the burdensome expenses associated with locating keys on the client's end and the considerable storage costs, we propose integrating a linearly-homomorphic encryption with preprocessing (LHEP), super-increasing sequence (SIS), secret sharing, Beaver's matrix multiplication, and bloom filters to construct a secure and efficient hint locating method. From a high-level viewpoint, our method enables the client to delegate the task of locating hints

Method 1: Secure and Efficient Hint Locating.

Algorithm $\texttt{KeyGen}\big((\mathbb{S}_i)_{i=1}^{\mathcal{M}}, \epsilon, k, K\big)$
  $(\mathbb{H}, m) \leftarrow \texttt{BF.Setup}(\epsilon, \sqrt{N}, k)$
  for $i = 1$ to $\mathcal{M}$ **do**
    $\mathbf{b}_i \leftarrow \texttt{BF.Build}(\mathbb{S}_i, \mathbb{H}, m)$
    $\mathbf{b}_i^0 \xleftarrow{\mathcal{R}} \{0,1\}^m$, $\mathbf{b}_i^1 \leftarrow \mathbf{b}_i \oplus \mathbf{b}_i^0$
  $\mathsf{dim}_0 \leftarrow (\mathcal{M}, n)$, $\mathsf{dim}_1 \leftarrow (n, K)$, $\mathsf{lsk} \xleftarrow{\mathcal{R}} \{0,1\}^\lambda$
  for $i = 1$ to K **do**
    $\mathbf{t}_i \leftarrow \texttt{Map}(\texttt{PRF}(\mathsf{lsk}, i)) \in \mathbb{Z}_q^n$
    $\mathbf{t}_i^0 \xleftarrow{\mathcal{R}} \mathbb{Z}_q^n$, $\mathbf{t}_i^1 \leftarrow \mathbf{t}_0 - \mathbf{t}_i^1 \bmod q$
  $(\mathbf{U}^0, \mathbf{U}^1, \mathbf{V}^0, \mathbf{V}^1, \mathbf{W}^0, \mathbf{W}^1) \leftarrow \texttt{BV.Gen}(\mathsf{dim}_0, \mathsf{dim}_1)$
  for $\beta = 0, 1$ **do**
    $\mathbb{B}_\beta \leftarrow (\mathbf{b}_i^\beta)_{i=1}^{\mathcal{M}}$, $\mathbb{T}_\beta \leftarrow (\mathbf{t}_i^\beta)_{i=1}^{K}$
  $\mathsf{pp} \leftarrow \texttt{LHEP.Setup}(\lambda)$
  return $(\mathsf{lsk}, \mathsf{pp}, \mathbb{H}, m, \mathbb{B}_0, \mathbb{B}_1, \mathbf{U}^0, \mathbf{U}^1, \mathbf{V}^0, \mathbf{V}^1, \mathbf{W}^0, \mathbf{W}^1)$
Algorithm $\texttt{KeyPrep}(\mathsf{pp}, \mathbb{B}_\beta, \mathbb{T}_\beta, \mathbf{U}^\beta, \mathbf{V}^\beta)$
  Parse $\mathbb{B}_\beta$ and $\mathbb{T}_\beta$ as $(\mathbf{b}_i^\beta)_{i=1}^{\mathcal{M}}$ and $(\mathbf{t}_i^\beta)_{i=1}^{K}$
  Initialize matrices $\mathbf{P}^\beta \leftarrow \{0\}^{\mathcal{M} \times n}$ and $\mathbf{T}^\beta \leftarrow \{0\}^{n \times K}$
  for $i = 1$ to $\mathcal{M}$ **do**
    $\mathbf{p}_i^\beta \leftarrow \texttt{LHEP.Prep}(\mathsf{pp}, \mathbf{b}_i^\beta)$
    Append row vector $\mathbf{p}_i^\beta$ into $\mathbf{P}^\beta$
  for $i = 1$ to K **do**
    Append column vector $\mathbf{t}_i^\beta$ into $\mathbf{T}^\beta$
  $(\mathbf{E}_0^\beta, \mathbf{E}_1^\beta) \leftarrow \texttt{BV.Prep}(\mathbf{P}^\beta, \mathbf{T}^\beta, \mathbf{U}^\beta, \mathbf{V}^\beta)$
  return $(\mathbf{E}_0^\beta, \mathbf{E}_1^\beta)$
Algorithm $\texttt{PrepProc}(\mathbf{U}^\beta, \mathbf{V}^\beta, \mathbf{E}_0^\beta, \mathbf{E}_1^\beta, \mathbf{E}_0^{\bar{\beta}}, \mathbf{E}_1^{\bar{\beta}}, \mathbf{W}^\beta)$
  return $\mathbf{F}^\beta \leftarrow \texttt{BV.Proc}(\mathbf{U}^\beta, \mathbf{V}^\beta, \mathbf{E}_0^\beta, \mathbf{E}_1^\beta, \mathbf{E}_0^{\bar{\beta}}, \mathbf{E}_1^{\bar{\beta}}, \mathbf{W}^\beta)$
Algorithm $\texttt{KeyQuery}(\mathsf{pp}, \mathsf{lsk}, \mathbb{H}, m, k, \theta, \xi)$
  Parse $\mathbb{H}$ as $(h_1, h_2, \ldots, h_k)$
  $\mathbf{t} \leftarrow \texttt{Map}(\texttt{PRF}(\mathsf{lsk}, \xi)) \in \mathbb{Z}_q^{\mathsf{n}}$
  $\mathbf{v} \leftarrow \{0\}^m$, $\mathbf{a} \leftarrow \texttt{SIS.Gen}(k, 2)$
  for $i = 1$ to k **do**
    $\mathbf{v}[h_i(\theta)] = \mathbf{a}[i]$
  $\mathbf{c} \leftarrow \texttt{LHEP.Enc}(\mathsf{pp}, \mathbf{t}, \mathbf{v})$
  return $(\mathbf{c}, \mathbf{a})$
Algorithm $\texttt{KeyAnswer}(\mathsf{pp}, \mathbf{c}, \mathbb{B}_\beta, \mathbf{F}^\beta, \xi)$
  Parse $\mathbb{B}_\beta$ as $(\mathbf{b}_i^\beta)_{i=1}^{\mathcal{M}}$
  Initialize an empty vector $\mathbf{q}^\beta \in \{0\}^{\mathcal{M}} \in \mathbb{Z}_q^{\mathcal{M}}$
  for $i = 1$ to $\mathcal{M}$ **do**
    $\mathbf{q}^\beta[i] \leftarrow \texttt{LHEP.Apply}(\mathsf{pp}, \mathbf{b}_i^\beta, \mathbf{c})$
  return $(\mathbf{q}^\beta, \hbar^\beta \leftarrow \mathbf{F}^\beta[:, \xi])$
Algorithm $\texttt{KeyLocate}(\mathsf{pp}, k, \mathbf{q}^0, \mathbf{q}^1, \hbar^0, \hbar^1, \mathbf{a})$
  $\mathbf{q} \leftarrow \mathbf{q}^0 + \mathbf{q}^1 \bmod q$, $\hbar \leftarrow \hbar^0 + \hbar^1 \bmod q$
  for $i = 1$ to $\mathcal{M}$ **do**
    $d_i \leftarrow \frac{\texttt{Round}_\Delta((\mathbf{q}[i] - \hbar[i]) \bmod q)}{\Delta} \bmod p$
    if $d_i = \sum_{i=1}^k \mathbf{a}[i]$ **then**
      return i

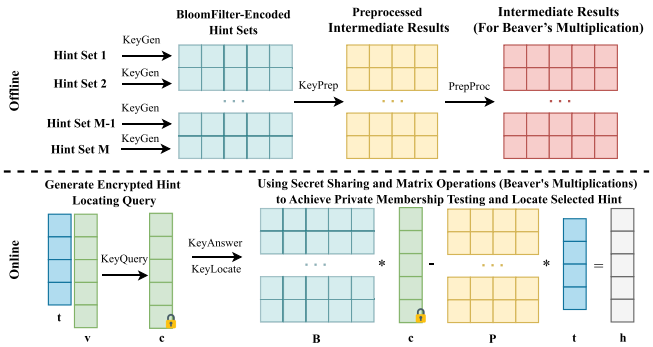


Fig. 3. High-level overview of secure hint locating. The offline phase builds Bloom filters and prepares LHEP/Beaver matrix triples. The online phase evaluates a private membership test as a matrix–vector computation and recovers the selected hint index.

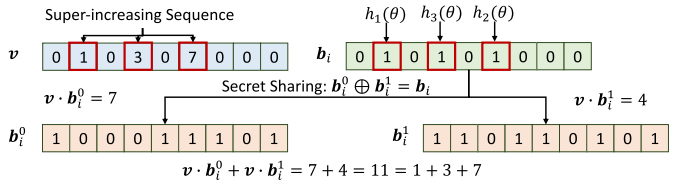


Fig. 4. An example of converting membership testing in a bloom filter to a dot product using super-increasing sequences.

to the servers. For efficiency, the hints are encoded as bloom filters before delegation, exploiting the fast membership testing capability. The client then interacts with the servers to securely locate the requisite hint. To ensure security, LHEP and secret sharing are employed to prevent the servers from extracting the query index θ based on the client's hint locating query. While the overarching idea is clear, there exist two challenges in designing the method.

The first problem is how to delegate the task of locating hints securely. In PIRS, each hint (sk_i, ζ_i) correlates to a set $\mathbb{S}_i$ of $\sqrt{N}$ indices, and the client encodes the set $\mathbb{S}_i$ to form an m-bit binary vector $\mathbf{b}_i = \texttt{BF.Build}(\mathbb{S}_i, \mathbb{H}, m)$ (i.e., a bloom filter). The secret shares of $\mathbf{b}_i$, $\mathbf{b}_i^0$ and $\mathbf{b}_i^1$, are outsourced to DS_0 and DS_1, respectively. With the secret shares, the client aims to determine whether the query index θ belongs to the set $\mathbb{S}_i$, based on the membership testing performed on $\mathbf{b}_i$. Note that, the membership testing is achieved by mapping the query index θ into k resulting positions using hash functions $\mathbb{H}$ and checking the values of resulting positions in the binary $\mathbf{b}_i$. Therefore, to support secure membership testing on the secret shares, the client constructs a secure hint locating query $c = \texttt{LHEP.Enc}(\mathsf{pp}, \mathbf{t}, \mathbf{v})$ using a secret key $\mathbf{t}$, which represents the encrypted form of a plaintext vector $\mathbf{v}$, and only the resulting positions (i.e., $h_1(\theta), h_2(\theta), \ldots, h_k(\theta)$) in $\mathbf{v}$ are populated with the values from an SIS $\mathbf{a}$ and other positions remain zero. In this way, the membership testing operation $\texttt{BF.Check}(\theta, \mathbf{b}_i)$ on the client's end gets converted into a dot product operation $\texttt{LHEP.Apply}(\mathsf{pp}, \mathbf{b}_i^\beta, \mathbf{c})$ on the server side. For easy understanding, we provide a simple example of the conversion procedure in Fig. 4. Finally, the client compares the decryption result d_i with the summation of the SIS $\mathbf{a}$ to find the needed hint's index δ.

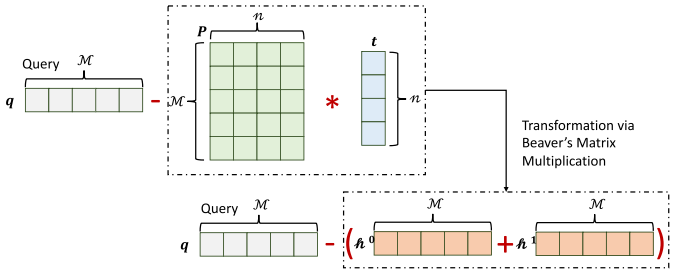


Fig. 5. The conceptual illustration of utilizing Beaver's matrix triples to transform the computation of $\mathbf{P} \cdot \mathbf{t}$ to the computation of $\mathbf{h}^0 + \mathbf{h}^1$.

The second problem is how to reduce communication costs or local storage overheads on the client's end when applying LHEP. In the original LHEP (as described in Section III-G), the client needs three components $(\mathsf{ct}, \mathbf{p}, \mathbf{t})$ to decrypt correctly. In PIRS, $\mathbf{p}$ corresponds to a matrix $\mathbf{P} = \mathbf{P}^0 + \mathbf{P}^1$ of size $\mathcal{M} \times n$, and the client has to download it in advance and store it locally. However, $\mathbf{P}$ is large, demanding the client reserves an extra storage of $\mathcal{O}(\mathcal{M} \cdot n \cdot \log_2 q)$. To overcome the challenge, Beaver's matrix multiplication is introduced. The client offloads partial decryption, i.e., the computation of $\mathbf{p} \cdot \mathbf{t}$ (In PIRS, the computation is $\mathbf{P} \cdot \mathbf{t}_i$, and $\mathbf{t}_i$ is a random secret key generated from PRF and a function $\mathtt{Map}(\cdot)$ that maps a random λ-bit binary to a pseudorandom group element of $\mathbb{Z}_q^n$) to the servers, and hence the client only needs to download two secret shares (vectors) $\hbar^0$ and $\hbar^1$ of size $\mathcal{M}$, which are much smaller than $\mathbf{P}^0$ and $\mathbf{P}^1$. The conceptual illustration of the procedure is given in Fig. 5. To achieve this, the client generates Beaver matrix triples $(\mathbf{U}^0, \mathbf{U}^1, \mathbf{V}^0, \mathbf{V}^1, \mathbf{W}^0, \mathbf{W}^1)$ and distributes them to the servers during the offline phase. By using homomorphic encryption and oblivious transfer, the Beaver matrix triples can also be generated without the client's participating [30]. With the Beaver matrix triples $(\mathbf{U}^\beta, \mathbf{V}^\beta, \mathbf{W}^\beta)$ and intermediates $(\mathbf{E}_0^\beta, \mathbf{E}_1^\beta, \mathbf{E}_0^{\bar{\beta}}, \mathbf{E}_1^{\bar{\beta}})$, DS_β pre-computes a matrix $\mathbf{F}^\beta$, which is for answering up to K secure hint locating queries from the client. The client maintains a counter $\xi \in [1, K]$ to track the usage of Beaver matrix triples because they cannot be used repeatedly. Each hint locating query depletes one Beaver matrix triple, but the client can regenerate them during the offline phase. According to the counter ξ, DS_0 and DS_1 respond $\hbar^0 = \mathbf{F}^0[:, \xi]$ and $\hbar^1 = \mathbf{F}^1[:, \xi]$, satisfying $\hbar = \hbar^0 + \hbar^1 = \mathbf{P} \cdot \mathbf{t}_\xi$, and the decryption procedure becomes

$$d_i = \frac{\mathtt{Round}_\Delta((\mathbf{q}[i] - \hbar[i]) \bmod q)}{\Delta} \bmod p. \tag{4}$$

Fig. 3 provides a high-level overview of secure hint locating. The detailed procedures of the proposed method are outlined in Method 1, under the algorithms $\mathtt{KeyGen}(\cdot)$, $\mathtt{KeyPrep}(\cdot)$, $\mathtt{PrepProc}(\cdot)$, and $\mathtt{KeyQuery}(\cdot)$. Finally, the client that has made PIR queries needs to refresh the outsourced key table by re-running $\mathtt{KeyPrep}(\cdot)$. It is necessary because some PIR keys have been refreshed during executing the algorithm $\mathtt{HtUpd}(\cdot)$. The operations should be done offline to speed up the online query efficiency.

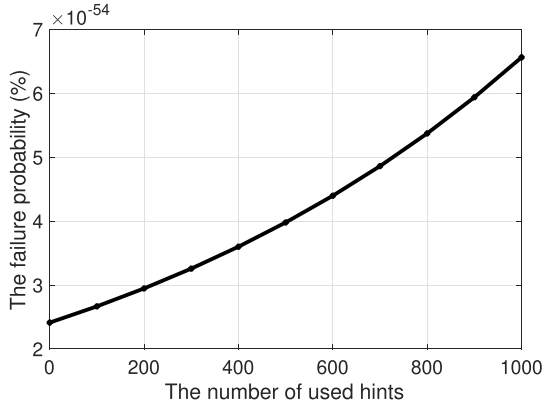


Fig. 6. The failure probability of the first case with changes in the number of used hints, given $N = 10^6$ and $\mathcal{M} = 128\sqrt{N}$.

V. Correctness and Security Analysis

A. Correctness Analysis

Theorem 1 (Correctness): Given BF parameters $(\epsilon, \sqrt{N}, k)$, LHEP parameters $\mathsf{pp} = \{n, q, p, \chi, m, \Delta, \mathbf{A}\}$, the plaintext modulus p, the size of hints $\mathcal{M} = \lambda\sqrt{N}$, for every database $\mathbb{DB} = (i, \mathsf{cnt}_i)_{i=1}^N$ with N records, the probability that a client succeeds in retrieving cnt_θ is overwhelming in PIRS.

Proof: There are three cases under which the success of PIR queries may be affected. The first case is that a client cannot find $\delta \in [1, \mathcal{M}]$ satisfying $\theta \in \mathtt{PPRS.Eval}_{\mathrm{II}}(\mathsf{sk}_\delta, \zeta_\delta)$. The probability of the first failure case occurring is identified as $Pr[\mathrm{Fail}] = (1 - 1/\sqrt{N})^{\mathcal{M}}$. As shown in Fig. 6, for continuous PIR queries without hint refreshing, the failure probability will increase, but the probability can be still negligible if the hints are large enough.

In the second case, we consider the impact of false positives introduced by BF encoding puncturable sets. The false positives (recalling (1)) can lead a client to mistakenly locate incorrect hints. The errors do not result in the failure of PIRS, since a client can locally verify the PIR keys.

The third case arises from the utilization of LHEP-based hint locating: if the LHEP decryption cannot be done correctly, the PIR query fails. Concretely, the core step of the decryption procedure works as follows:

$$\begin{aligned}
&(\mathbf{q}^0[i] + \mathbf{q}^1[i]) - (\mathbf{h}^0 + \mathbf{h}^1) \\
&= (\mathbf{q}^0[i] + \mathbf{q}^1[i]) - ((\mathbf{P}^0[i:0] + \mathbf{P}^1[i:0]) \cdot \mathbf{t}_i) \\
&= (\mathbf{b}_i^0 + \mathbf{b}_i^1) \cdot \mathbf{c} - (\mathbf{b}_i^0 + \mathbf{b}_i^1) \cdot \mathbf{A} \cdot \mathbf{t}_i \\
&= \mathbf{b}_i \cdot (\mathbf{A} \cdot \mathbf{t}_i + \mathbf{e} + \Delta \cdot \mathbf{v}) - \mathbf{b}_i \cdot \mathbf{A} \cdot \mathbf{t}_i \\
&= \mathbf{b}_i \cdot \mathbf{e} - \mathbf{b}_i \cdot \Delta \cdot \mathbf{v}.
\end{aligned} \tag{5}$$

The decryption proceeds correctly when $|\mathbf{b}_i \cdot \mathbf{e}| < \Delta/2$, which implies $Pr[\mathrm{Fail}] = Pr[|\mathbf{b}_i \cdot \mathbf{e}| \geq \Delta/2]$. According to Appendix C.2 in [12], given a discrete Gaussian distribution χ ($\mathbf{e} \in \chi^m$,

and $m = m$), we can obtain

$$|\,Pr[|\mathbf{b}_i \cdot \mathbf{e}| \geq \Delta/2]| < \mu$$

as long as

$$2\exp\left(-\pi \cdot \left(\frac{\Delta}{2\sqrt{2\pi}\sigma \cdot \|\mathbf{b}_i\|}\right)^2\right) \leq \mu.$$

Equivalently, considering that $\|\mathbf{b}_i\| \leq \sqrt{m \cdot 2^2} = 2\sqrt{m}$, we have

$$\lfloor q/p \rfloor \geq 4\sqrt{2\pi m} \cdot \sigma \cdot \sqrt{-\frac{1}{\pi}\ln\left(\frac{\mu}{2}\right)}.$$

In summary, by choosing appropriate values for λ, ϵ, and μ, we can guarantee the correctness of PIRS. □

B. Security Analysis

Using a simulation-based approach, we formally demonstrate that PIRS is $\mathcal{L}-$adaptively secure based on certain foundational primitives. The goal of the security analysis is to construct a simulator, $\mathcal{S}$, that can use permitted leakages (captured by leakage functions) to simulate an ideal experiment for a PPT adversary, $\mathcal{A}$ that controls DS_β where $\beta = 0$ or 1. The adversary cannot computationally distinguish the ideal experiment from a real experiment.

Leakage Functions: The leakage functions are defined as follows: $\mathcal{L}^{\text{hprep}}(\cdot) = (\mathsf{ts}^{\text{hprep}}, \mathbb{HT})$, $\mathcal{L}^{\text{qu}}(\cdot) = (\mathsf{ts}^{\text{qu}}, \delta)$, $\mathcal{L}^{\text{upd}}(\cdot) = (\mathsf{ts}^{\text{upd}}, \delta, \mathcal{L}^{\text{DPF}}(\cdot))$, $\mathcal{L}^{\text{kprep}}(\cdot) = (\mathsf{ts}^{\text{kprep}}, \mathcal{L}^{\text{BV}}(\cdot), \mathsf{pp})$, $\mathcal{L}^{\text{proc}}(\cdot) = (\mathsf{ts}^{\text{proc}})$, and $\mathcal{L}^{\text{kqu}}(\cdot) = (\mathsf{ts}^{\text{kqu}}, \mathbf{h}, k, m, \mathbf{a})$ where $\mathsf{ts}^{\text{hprep}}$, ts^{qu}, ts^{upd}, $\mathsf{ts}^{\text{kprep}}$, $\mathsf{ts}^{\text{kprep}}$, $\mathsf{ts}^{\text{proc}}$, and ts^{kqu} are the timestamps when a client initiates algorithm executions, and $\mathcal{L}^{\text{DPF}}(\cdot)$ is the leakage functions of DPF (see details in the following proof). Here, in addition to original leakage functions defined in Section II-C, four more leakage functions are introduced to capture what information is leaked to $\mathcal{A}$ during the executions of $\texttt{KeyPrep}(\cdot)$, $\texttt{PrepProc}(\cdot)$, and $\texttt{KeyQuery}(\cdot)$.

Theorem 2: PIRS achieves $\mathcal{L}$-adaptive security.

Proof: The security proof is based on a sequence of games. The game starts from the real experiment, and ends with the ideal experiment. We argue that for every two consecutive games, the probability that a PPT adversary $\mathcal{A}$ can differentiate between them is negligible.

Game 0: The start-point game G_0 is the same as the real experiment, $\mathsf{REAL}_{\mathcal{A}}^{\text{PIRS}}(\lambda)$ such that

$$Pr[\mathsf{REAL}_{\mathcal{A}}^{\text{PIRS}}(\lambda) = 1] = Pr[G_0 = 1].$$

Game 1: G_1 is identical to G_0 except that $\mathcal{S}$ utilizes the leakages of the point function (i.e., $\mathcal{L}^{\text{DPF}}(f)$ where $f(\cdot) : \rho \to y^*$) to construct a DPF simulator, and replaces the function call to $\texttt{DPF.Gen}(\cdot)$ by the DPF simulator $\texttt{SIM}(\mathcal{L}^{\text{DPF}}(f))$. If $\mathcal{A}$ can distinguish between G_1 and G_0, then it becomes feasible to construct an adversary $\mathcal{B}_0$ that challenges the simulation security of DPF. The distinction is bounded by:

$$|\,Pr[G_1 = 1] - Pr[G_0 = 1]| \leq \mathsf{Adv}_{\text{DPF},\mathcal{B}_0}^{\text{Sim}}(\lambda).$$

Game 2: In this game, $\mathcal{S}$ introduces randomly selected values from their respective domains to substitute secret-shared values, including PW_i^β, $\mathsf{PW}_{\text{upd}}^\beta$, $\mathbf{b}_i^\beta$, $\mathbf{t}_i^\beta$, and $\mathbf{b}^\beta$. Owing to the information-theoretic security of secret sharing and the random permutation $\phi(\cdot)$, formally we have

$$Pr[G_2 = 1] = Pr[G_1 = 1].$$

Game 3: The difference between G_3 and G_2 is that $\mathcal{S}$ uses a random element $\mathbf{t}$ selected from $\mathbb{Z}_q^n$. In this case, $\mathcal{A}$ cannot distinguish G_3 and G_2 with non-negligible probability, otherwise, an adversary $\mathcal{B}_1$ can be constructed to break the pseudorandomness of PRF, that is, we have

$$|\,Pr[G_3 = 1] - Pr[G_2 = 1]| \leq \mathsf{Adv}_{\text{PRF},\mathcal{B}_1}^{\text{Prand}}(\lambda).$$

Game 4: The difference between G_4 and G_3 is that $\mathcal{S}$ computes an encrypted constant zero vector using a random secret key $\mathbf{t}$ as $\mathbf{c} = \texttt{LHEP.Enc}(\mathsf{pp}, \mathbf{t}, \mathbf{0})$ and subsequently replaces $\mathbf{c} = \texttt{LHEP.Enc}(\mathsf{pp}, \mathbf{t}, \mathbf{v})$. If $\mathcal{A}$ can discern a distinction between G_4 and G_3, then it is possible to construct an adversary $\mathcal{B}_2$ that breaks the security of LHEP. Therefore, we have

$$|\,Pr[G_4 = 1] - Pr[G_3 = 1]| \leq \mathsf{Adv}_{\text{LHEP},\mathcal{B}_2}^{\text{Ind}}(\lambda).$$

Game 5: The difference between G_5 and G_4 is that $\mathcal{S}$ utilizes the simulator of Beaver's matrix multiplication $\texttt{SIM}(\mathcal{L}^{\text{DPF}}(\mathbf{P}, \mathbf{T}))$ to replace the function calls to $\texttt{BV.Gen}(\cdot)$ and $\texttt{BV.Proc}(\cdot)$. If $\mathcal{A}$ can distinguish between G_5 and G_4, then an adversary $\mathcal{B}_3$ can utilize the difference to beak the simulation security of Beaver's matrix multiplication. Hence, we have:

$$|\,Pr[G_5 = 1] - Pr[G_4 = 1]| \leq \mathsf{Adv}_{\text{BV},\mathcal{B}_3}^{\text{Sim}}(\lambda).$$

Game 6: G_6 is identical to G_5 except that $\mathcal{S}$ additionally replaces the extra index r_β in the PIR query with a random index $r_\beta \in \mathbb{S}_\beta \backslash \{\varkappa_\beta\}$. From the view of $\mathcal{A}$, G_6 and G_5 are identical, since r_β is originally a random element chosen from the punctured set $\mathbb{S}_\beta \backslash \{\varkappa_\beta\}$, i.e.,

$$Pr[G_6 = 1] = Pr[G_5 = 1]$$

Game 7: G_7 makes slight changes on top of G_6. In the game, $\mathcal{S}$ sets $\mathsf{sk}_1^* = \mathsf{sk}_0^* = \texttt{PPRS.Punc}(\mathsf{sk}_{\text{new}}, \zeta_{\text{new}}, \varkappa_0)$ and $\zeta_1^* = \zeta_0^* = \zeta_{\text{new}}$, instead of deriving $(\mathsf{sk}_1^*, \zeta_1^*)$ from puncturing the key sk_δ and the offset ζ_δ. Furthermore, $\mathcal{S}$ removes the steps of randomly picking ρ and checking $\rho \stackrel{?}{=} \delta$, and updating sk_1' and ζ_1'. There are two cases:

- ($\mathcal{A}$ controls DS_0) $\mathcal{A}$ cannot distinguish G_7 and G_6, as the modification only affects $(\mathsf{sk}_1^*, \zeta_1^*)$ and $(\mathsf{sk}_1', \zeta_1')$ rather than $(\mathsf{sk}_0^*, \zeta_0^*)$ and $(\mathsf{sk}_0', \zeta_0')$.
- ($\mathcal{A}$ controls DS_1) In this case, the difference between G_7 and G_6 includes two parts: 1) sk_1 and ζ_1 are replaced by $\mathsf{sk}_{\text{new}} = \texttt{PPRS.Gen}(\lambda, N)$ and ζ_{new}, and the index θ is punctured; 2) sk_1' and ζ_1' are fixed as a newly generated puncturable-set key and a random offset. We can observe that the distributions of the original $(\mathsf{sk}_1^*, \zeta_1^*)$ and $(\mathsf{sk}_0^*, \zeta_0^*)$ are identical, and the distributions of the newly generated $(\mathsf{sk}_1', \zeta_1')$ and $(\mathsf{sk}_\rho, \zeta_\rho)$ are identical. If $\mathcal{A}$ can distinguish them with non-negligible advantage, an adversary $\mathcal{B}_4$ can be constructed to break the pseudorandomness of PPRS.

Hence, we have

$$|\,Pr[G_7 = 1] - Pr[G_6 = 1]| \leq \mathsf{Adv}^{\mathrm{Prand}}_{\mathrm{PPRS},\mathcal{B}_4}(\lambda).$$

Game 8: The deviation in G_8 from G_7 is characterized by a modification where $\mathcal{S}$ takes the query index θ to a random index $\theta \in [1, N]$. In the situation, if $\mathcal{A}$ can distinguish G_8 and G_7, we can establish an adversary $\mathcal{B}_5$ to break the security of PPRS. Then, we have

$$|\,Pr[G_8 = 1] - Pr[G_7 = 1]| \leq \mathsf{Adv}^{\mathrm{Ind}}_{\mathrm{PPRS},\mathcal{B}_5}(\lambda).$$

Game 9: In this game, $\mathcal{S}$ alters the sampling process for α, setting its value to $\frac{\sqrt{N}-1}{N}$, and fixes γ to $\bar{\beta}$. Considering the modifications made in the preceding games, when $\gamma = \bar{\beta}$, the PIR query generated for $\alpha = 1$ aligns with the query for $\alpha = 0$. This implies that $\mathcal{S}$ can make such a simple transposition without affecting the view of $\mathcal{A}$. So we have

$$Pr[G_9 = 1] = Pr[G_8 = 1].$$

Game 10: In the final game, $\mathcal{S}$ replaces $\mathsf{PW}_{\mathrm{new}}$ with a random element chosen from $\{0,1\}^{\ell}$. If $\mathcal{A}$ cannot distinguish G_{10} and G_9 with non-negligible probability, an adversary $\mathcal{B}_6$ can be constructed to break the pseudorandomness of PRF, that is, we have

$$|\,Pr[G_{10} = 1] - Pr[G_9 = 1]| \leq \mathsf{Adv}^{\mathrm{Prand}}_{\mathrm{PRF},\mathcal{B}_6}(\lambda).$$

We give Game 10 explicitly in Fig. 7. Based on Lemma 36 in [12], $\mathcal{A}$ cannot computationally distinguish two PIR queries with two different query indices in the game, and other elements sent to $\mathcal{A}$ are independent of θ. Together with the indistinguishability of the sequence of games, $\mathcal{S}$ simulates the ideal experiment eventually, and we have

$$\Pr[G_{10} = 1] = \Pr[\mathsf{IDEAL}^{\mathrm{PIRS}}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)] = 1$$

and

$$|\,Pr[\mathsf{REAL}^{\mathrm{PIRS}}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{IDEAL}^{\mathrm{PIRS}}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)]| \leq \mathtt{negl}(\lambda),$$

which completes the proof. □

HtPrep: Simulation of *Honest User*

Input: $\mathcal{L}^{\mathrm{hprep}}(\cdot)$
Output: $\mathbb{PH}_{\beta}$
for $i = 1$ to $\mathcal{M}$ **do**
  $\mathsf{PW}_i^{\beta} \xleftarrow{\mathcal{R}} \{0,1\}^{\ell}$
$\mathbb{PH}_{\beta} \leftarrow (\mathsf{PW}_i^{\beta})_{i=1}^{\mathcal{M}}$

KeyPrep: Simulation of *Honest User*

Input: $\mathcal{L}^{\mathrm{kprep}}(\cdot)$
Output: $(\mathbb{B}_{\beta}, \mathbb{T}_{\beta}, \mathbf{U}^{\beta}, \mathbf{V}^{\beta}, \mathbf{W}^{\beta})$
for $i = 1$ to $\mathcal{M}$ **do**
  $\mathbf{b}_i^{\beta} \xleftarrow{\mathcal{R}} \{0,1\}^m$
for $i = 1$ to K **do**
  $\mathbf{t}_i^{\beta} \xleftarrow{\mathcal{R}} \mathbb{Z}_q^n$
$\mathbb{B}_{\beta} \leftarrow (\mathbf{b}_i^{\beta})_{i=1}^{\mathcal{M}}$, $\mathbb{T}_{\beta} \leftarrow (\mathbf{t}_i^{\beta})_{i=1}^{K}$
$(\mathbf{U}^0, \mathbf{U}^1, \mathbf{V}^0, \mathbf{V}^1, \mathbf{W}^0, \mathbf{W}^1) \leftarrow \mathtt{SIM}(\mathcal{L}^{\mathrm{BV}}(\mathbf{P}, \mathbf{T}))$

ProcPrep: Simulation of *Honest* $DS_{\bar{\beta}}$

Input: $\mathcal{L}^{\mathrm{proc}}(\cdot)$
Output: $(\mathbf{E}_0^{\bar{\beta}}, \mathbf{E}_1^{\bar{\beta}})$
$(\mathbf{E}_0^{\bar{\beta}}, \mathbf{E}_1^{\bar{\beta}}) \leftarrow \mathtt{SIM}(\mathcal{L}^{\mathrm{BV}}(\mathbf{P}, \mathbf{T}))$

KeyQuery: Simulation of *Honest User*

Input: $\mathcal{L}^{\mathrm{kqu}}(\cdot)$
Output: $\mathbf{c}$
$r \xleftarrow{\mathcal{R}} \{0,1\}^{\lambda}$, $\mathbf{t} \leftarrow \mathtt{Map}(r)$
$\mathbf{c} \leftarrow \mathtt{LHEP.Enc}(\mathsf{pp}, \mathbf{t}, \{0\}^m)$

Query: Simulation of *Honest User*

Input: $\mathcal{L}^{\mathrm{qu}}(\cdot)$
Output: qu_{β}
$\theta \xleftarrow{\mathcal{R}} [1, N]$
Sample $\alpha \leftarrow \mathtt{Bernoulli}(\frac{\sqrt{N}-1}{N})$
$\mathsf{sk}_{\mathrm{new}} \leftarrow \mathtt{PPRS.Gen}(\lambda, N)$, $\zeta_{\mathrm{new}} \leftarrow \mathtt{PPRS.Loc}(\mathsf{sk}_{\mathrm{new}}, \theta)$
$\mathbb{S}_{\mathrm{new}} \leftarrow \mathtt{PPRS.Eval}_{\mathrm{II}}(\mathsf{sk}_{\mathrm{new}}, \zeta_{\mathrm{new}})$
$\mathsf{sk}'_{\beta} \leftarrow \mathtt{PPRS.Gen}(\lambda, N)$, $\zeta'_{\beta} \xleftarrow{\mathcal{R}} [1, N]$
if $\alpha = 0$ **then**
  $\varkappa_{\beta} \leftarrow \theta$, $\mathbb{S}_{\beta} \leftarrow \mathbb{S}_{\mathrm{new}}$, $r_{\beta} \xleftarrow{\mathcal{R}} \mathbb{S}_{\beta} \backslash \{\varkappa_{\beta}\}$
  $\mathsf{sk}^*_{\beta} \leftarrow \mathtt{PPRS.Punc}(\mathsf{sk}_{\mathrm{new}}, \zeta_{\mathrm{new}}, \varkappa_{\beta})$, $\zeta^*_{\beta} \leftarrow \zeta_{\mathrm{new}}$
else if $\alpha = 1$ **then**
  $\mathbb{S}_{\beta} \leftarrow \mathbb{S}_{\mathrm{new}}$, $\zeta^*_{\beta} \leftarrow \zeta_{\mathrm{new}}$, $\varkappa_{\beta} \xleftarrow{\mathcal{R}} \mathbb{S}_{\beta} \backslash \{\theta\}$, $r_{\beta} \leftarrow \mathbb{S}_{\beta} \backslash \{\varkappa_{\beta}\}$
  $\mathsf{sk}^*_{\beta} \leftarrow \mathtt{PPRS.Punc}(\mathsf{sk}_{\mathrm{new}}, \zeta_{\mathrm{new}}, \varkappa_{\beta})$
$\mathsf{qu}_{\beta} \leftarrow (\delta, \mathsf{sk}'_{\beta}, \zeta'_{\beta}, \mathsf{sk}^*_{\beta}, \zeta^*_{\beta}, r_{\beta})$

HtUpd: Simulation of *Honest User*

Input: $\mathcal{L}^{\mathrm{upd}}(\cdot)$
Output: rq_{β}
$(K_0, K_1) \leftarrow \mathtt{SIM}(\mathcal{L}^{\mathrm{DPF}}(f))$
$\mathsf{PW}^{\beta}_{\mathrm{upd}} \xleftarrow{\mathcal{R}} \{0,1\}^{\ell}$, $\mathsf{PW}_{\mathrm{new}} \xleftarrow{\mathcal{R}} \{0,1\}^{\ell}$
$\mathsf{rq}_{\beta} \leftarrow (\delta, K_{\beta}, \mathsf{PW}^{\beta}_{\mathrm{upd}}, \mathsf{PW}_{\mathrm{new}})$

Fig. 7. The construction of a simulator, $\mathcal{S}$, that simulates Game 10.

VI. Implementation & Evaluation

In our experiment, we configure the security parameter λ to be 128 to ensure robust security. The false positive rate is set to $\epsilon = 0.01$, and we select the number of hash functions $k = 7$ and the size of the hints $\mathcal{M} = \lambda\sqrt{N}$ with $\lambda = 128$. We set the size of version number $|\mathsf{ver}|$ to 20 bits. For LHEP, the parameters are chosen as follows: $n = 1024$, $q = 2^{32}$, and $p = 4096$.

Experimental Environments: The experiment is run in a MAC OS with M1 CPU and 16 GB RAM, and we omit the transmission delay. We implement the proof-of-concept prototype of PIRS using Python, and using *Sqlite3* to write/read databases. The Python-based prototype is single-threaded and can be extended to support multiple threads, which may proportionately boost computational efficiency to the number of available processor threads.

Complexity Analysis: In PIRS, the *offline* phase costs $O(\sqrt{N}(\lambda + \log N))$ for key generation and $O(N\ell)$ bitwise XOR for parity-word construction, with $O(\sqrt{N}\,\ell)$ communication to distribute the $O(\sqrt{N})$ parity words to the two servers. In the *online* phase, the server-side computation per query is sublinear, $O(\sqrt{N} \cdot \ell)$ bit-operations, i.e., $O(\sqrt{N})$ XORs each applied to a block of length ℓ, while the client performs only lightweight PRF evaluations. The per-query communication is $O(\lambda \log N + \ell)$, consisting of two PIR requests of size $O(\lambda \log N)$ and a downlink of size $O(\ell)$. Hint refresh can be executed lazily with amortized $O(\sqrt{N})$ server-side computation and negligible communication. For the server-assisted hint locating method, *offline* preprocessing requires $O(N)$ operations for Bloom-filter encoding and $O(\sqrt{N} \cdot \mathrm{poly}(\lambda))$ for matrix preparation. Each *online* lookup then incurs $O(\sqrt{N})$ server-side operations and

TABLE I
COMPARISON OF CLIENT STORAGE AND COMMUNICATION COSTS

Database Size	2^{16}	2^{18}	2^{20}
LinearScan (Storage)	589 KB	1.19 MB	2.42 MB
HashStore (Storage)	17.36 MB	76.69 MB	337.96 MB
Our Method (Storage)	671 KB	1.35 MB	2.75 MB
Our Method (Uplink)	9 KB	19 KB	39 KB
Our Method (Downlink)	131 KB	262 KB	524 KB
Our Method (Total)	140 KB	281 KB	563 KB

TABLE II
OFFLINE PROCESSING COSTS OF *HASHSTORE* AND OUR METHOD (UNIT: SECONDS)

Database Size	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
HashStore (Client)	0.07	0.48	3.08	16.72	353.64
`KeyGen` (Client)	1.74	5.63	20.28	75.26	304.39
`KeyPrep` (Server)	4.55	19.31	135.21	888.63	3544.71
`PrepProc` (Server)	0.69	1.39	2.81	5.61	11.23

$O(\sqrt{N})$ communication (the servers return the membership vector of length $M = \Theta(\sqrt{N})$), while the client computation remains negligible.

A. Hint Locating Efficiency

Since the proposed hint locating method can be regarded as an independent module of PIRS, we evaluate its performance solely, and compare it to the conventional methods: *LinearScan* and *HashStore* (described in Section IV-B).

Client Storage Costs & Communication Costs: Both our method and *LinearScan* necessitate that a client stores PIR Keys (sk_i, ζ_i) for $i = 1$ to $\mathcal{M}$, each pair requiring storage of size $\lambda + \log_2 N$ bits, leading to a total storage requirement of $\mathcal{M} \cdot (\lambda + \log_2 N)$ bits. Our method additionally needs to store a secret key vsk and $\mathcal{M}$ version numbers. *HashStore* requires a client to store not only PIR keys but also a hashtable, which contains $\mathcal{M} \cdot \sqrt{N}$ entries of $\log_2 N$ bits each, thus incurring significantly higher storage costs compared to our method and *LinearScan*. In *LinearScan* and *HashTable*, a client can locate the appropriate hint locally without interactions, while our method requires a client to interact with two servers. The uplink overheads of our method are $m \cdot \log_2 q$, and the downlink overheads are $\mathcal{M} \cdot \log_2 q$. We compare the client storage and communication costs in Table I. In short, our method and *LinearScan* have comparable storage costs, but *HashStore* demands more storage space, and our method requires additional server interactions.

Computational Costs of Hint Locating: Unlike *LinearScan*, both our method and *HashStore* use offline processing to speed up online hint locating, and *HashStore* is much more computationally efficient. Although the incorporation of LHEP and matrix multiplications in our offline processing phase leads to increased offline computational costs, as detailed in Table II, the performance gains our method offers are still evident: compared to *HashStore*, a client only needs to keep the KB-level storage costs instead of MB-level (recalling Table I). In the meantime, as shown in Fig. 8, it is clear that our method achieves higher online hint locating efficiency compared to *LinearScan*, taking around 610 ms to locate an appropriate hint even when the database size is 2^{20}. Also, Table III shows that most of the computational costs are delegated to the servers.

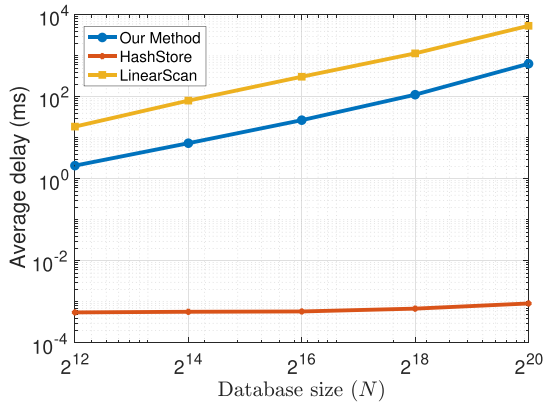


Fig. 8. The comparative performance of online hint locating delay: Our method vs. *HashStore* vs. *LinearScan*.

TABLE III
COMPUTATION COSTS OF HINT LOCATING ON CLIENT AND SERVER SIDES (UNIT: MILLISECONDS)

Database Size	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
Client	0.4	0.8	1.6	3.5	33
Server	1.7	6.6	26	110	610

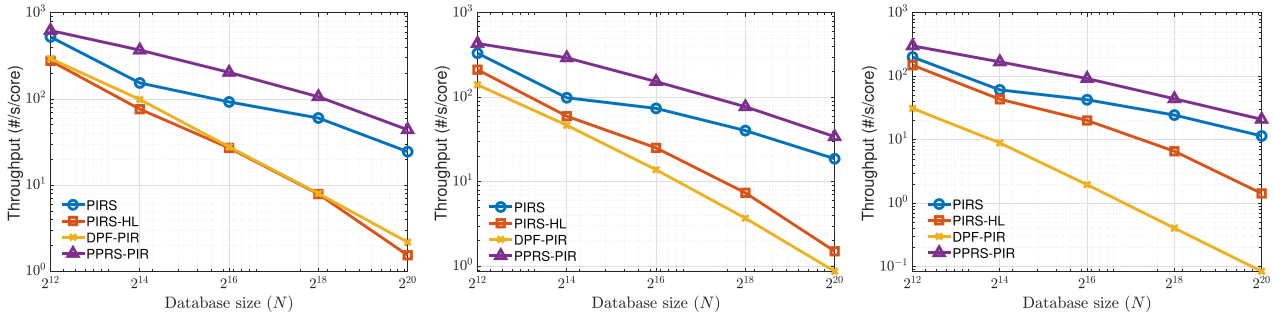
B. Performance Evaluation of PIRS

Database Configurations: We adopt database sizes for our experiment as 2^{12}, 2^{14}, 2^{16}, 2^{18}, and 2^{20} , and set the size of each database record as 0.1 KB, 1 KB, and 10 KB. These settings were employed to simulate different application scenarios. When the database contains $2^{20} = 1,048,576$ records, each record being 10 KB in size, it culminates in a total database size of 10.485 GB.

Benchmarks: We benchmark PIRS against two classical two-server PIR schemes, DPF-PIR and PPRS-PIR, in terms of computational, communication, and storage overheads. PIRS is implemented in two variants: **PIRS** and **PIRS-HL**. The key distinction between them lies in whether to apply the proposed hint locating method. In PIRS, clients maintain a reverse lookup table to accomplish hint locating locally. PIRS-HL employs the proposed hint locating method without storing such a table. The two state-of-the-art schemes are summarized as follows:

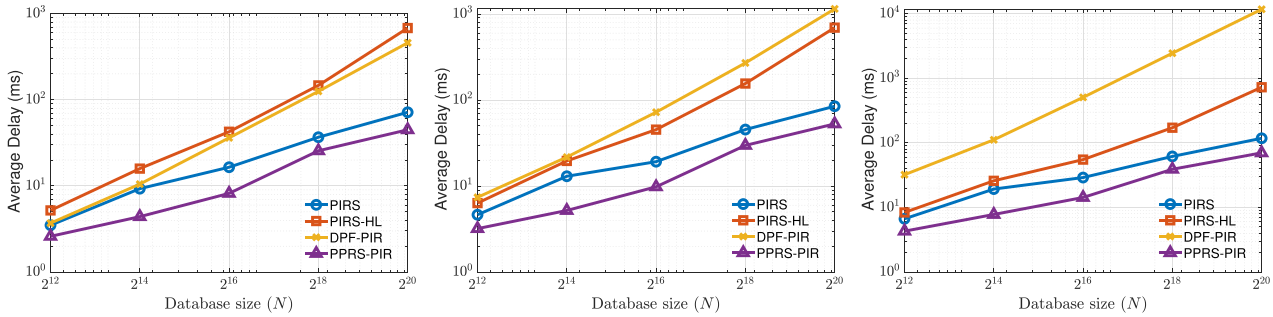
- *DPF-PIR [28]:* This is an efficient construction of DPF, which can be used to realize efficient PIR under two-server models.
- *PPRS-PIR [13]:* This is one of the most classical two-server offline/online private information retrieval schemes constructed using PPRS.

One-Time Costs of Hint Generation: PPRS-PIR and PIRS incur similar one-time hint initiation costs, except that PIRS needs an additional step: the secure distribution of generated



(a) Database Record Size 0.1 KB (b) Database Record Size 1 KB (c) Database Record Size 10 KB

Fig. 9. Average server throughput of PIRS, DPF-PIR, and PPRS-PIR with different database sizes N and record sizes ℓ.



(a) Database Record Size 0.1 KB (b) Database Record Size 1 KB (c) Database Record Size 10 KB

Fig. 10. Average end-to-end computational delay of PIRS, DPF-PIR, and PPRS-PIR with different database sizes N and record sizes ℓ.

hints across two servers. Utilizing *Sqlite3* for database interactions, our experimental setup indicates that hint generation and outsourcing take several minutes to hours, contingent on the size of the database size N and the record size ℓ. The procedure is required only once.

Server Throughput & End-to-End Delay: Figs. 9 and 10 show the server throughput and average computational costs for PIRS, PIRS-HL, DPF-PIR, and PPRS-PIR across three different record sizes ℓ. Evaluating different record sizes is necessary because ℓ directly affects the amount of database reading and XOR operations, which dominate per-query cost and thus influence both throughput and end-to-end delay. Notably, PIRS and PIRS-HL consistently outperform DPF-PIR as the sizes of the databases and records increase. This improvement results from PIRS's ability to achieve sublinear online computational complexity through strategic preprocessing. Moreover, across subfigures (a)–(c), PIRS, PIRS-HL, and PPRS-PIR exhibit similar trends, indicating that their performance is relatively insensitive to record size. By contrast, DPF-PIR shows much sharper degradation with larger ℓ, since each query must read all N records and perform $O(N)$ XOR operations over size-ℓ data, while the other schemes require only $O(\sqrt{N})$ such operations. Within our schemes, PIRS-HL shows a minor increase in computational costs compared to PIRS, due to additional client-server interactions for hint locating. Furthermore, the computational expenses for all schemes increase with record size, which is linked to the higher costs of *Sqlite3* database access and XOR operations when processing larger records. While PPRS-PIR offers greater computational efficiency than PIRS and PIRS-HL by eliminating the need for extra calculations related to hint refreshing, PIRS and PIRS-HL significantly reduce the storage burden on the client, as detailed in the subsequent analysis.

Client Storage Costs & Communication Overheads: Table IV presents a comparison of client storage and communication overheads for two distinct database configurations. Notably, PPRS-PIR requires substantially more storage than both PIRS and PIRS-HL. This increase in storage is primarily due to the need to retain partial hints $(\mathsf{PW}_i)_{i=1}^{\mathcal{M}}$, which grow with both the database size N and the record size. PIRS additionally demands extra storage to store a hashtable for hint locating purposes. While DPF-PIR eliminates the need for client storage, it places extra computational load on the server that is directly proportional to the database size. The impact of database size on storage is particularly evident in PPRS-PIR and PIRS: as the database size expands from 2^{16} to 2^{20}, the required storage rapidly increases from megabytes to gigabytes in PPRS-PIR. In contrast, PIRS-HL demonstrates a remarkable storage efficiency, where the storage overhead significantly decreases from megabytes to kilobytes. This observation underscores PIRS-HL's advantages in storage efficiency across various database sizes. Due to the outsourcing of hints, PIRS and PIRS-HL incur higher communication costs compared to DPF-PIR and PPRS-PIR.

TABLE IV
CLIENT STORAGE AND COMMUNICATION OVERHEADS OF PIRS, DPF-PIR, AND PPRS-PIR WITH DIFFERENT DATABASE SIZES N AND RECORD SIZES ℓ

Database Configurations	$N = 2^{16}, \ell = 80000$ (10 KB)				$N = 2^{20}, \ell = 80000$ (10 KB)			
Schemes	DPF-PIR	PPRS-PIR	PIRS	PIRS-HL	DPF-PIR	PPRS-PIR	PIRS	PIRS-HL
Storage	N/A	345.04 MB	17.36 MB	671 KB	N/A	1.64 GB	337.96 MB	2.75 MB
Uplink	356 Bytes	264 Bytes	1.1 KB	10.1 KB	613 Bytes	330 Bytes	1.2 KB	39.2 KB
Downlink	20 KB	40 KB	80 KB	211 KB	20 KB	40 KB	80 KB	604 KB

TABLE V
COMPUTATIONAL AND COMMUNICATION OVERHEADS OF HINT REFRESHING OVER DATABASES WITH RECORD SIZE 10 KB

Database Size	2^{14}	2^{16}	2^{18}	2^{20}
`HtUpd` (Client)	0.38 ms	0.41 ms	0.45 ms	0.54 ms
`HtRfsh` (Server)	3.9 ms	8.1 ms	13.9 ms	29.7 ms
Uplink	20.32 KB	20.35 KB	20.39 KB	20.42 KB

TABLE VI
CLIENT STORAGE COMPARISON OF PPRS-PIR, PIRS, AND PIRS-HL (UNIT: MEGABYTES)

Database Size	N	PPRS-PIR	PIRS	PIRS-HL
40	2^{12}	82.85	0.93	0.16
160	2^{14}	167.80	3.96	0.33
655	2^{16}	345.04	17.36	0.67
2621	2^{18}	732.05	76.69	1.35
10485	2^{20}	1648.69	337.96	2.75

Outsourced Hint Refreshing Costs: Table V shows the computational and communication overheads associated with outsourced hint refreshing. The hint update time (`HtUpd`) on the client's end takes less than 0.5 ms. The server-side hint refreshing (`HtRfsh`) is more time-consuming, with 29.7 ms needed for the large database and 1.7 ms for the smaller ones. The communication overheads correlate with the size of the databases.

Deployment Implications: Table VI compares the client storage of PPRS-PIR, PIRS, and PIRS-HL with the record size fixed to $\ell = 10$ KB. The results show that the client storage requirement of PPRS-PIR grows rapidly as the database size increases, while PIRS eliminates the dependence on ℓ and PIRS-HL further reduces the storage cost substantially. For instance, with a 655 MB database ($N = 2^{16}$), the client storage of PPRS-PIR is 345 MB, whereas PIRS and PIRS-HL require only 17 MB and 0.67 MB, respectively. At 2.62 GB ($N = 2^{18}$), the client storage of PPRS-PIR reaches 732 MB, compared to 77 MB for PIRS and 1.36 MB for PIRS-HL. At 10.49 GB ($N = 2^{20}$), the gap becomes even more pronounced: the client storage of PPRS-PIR reaches 1,649 MB, while PIRS remains at 338 MB and PIRS-HL still only 2.75 MB. These results indicate that PPRS-PIR is acceptable for small databases (below a few hundred MB), but once the database size exceeds approximately 500 MB, PIRS and especially PIRS-HL are recommended for deployment due to their substantially improved scalability.

VII. RELATED WORK

As our work focuses on two-server offline/online PIR, in this section, we mainly review the state-of-the-art PIR schemes under the two-server model, and especially with preprocessing.

Two-Server PIR with Linear Complexity: The earliest two-server PIR scheme was proposed by Chor et al. [11]. Under the non-colluding assumption where two servers store identical copies of an N-entry database, their scheme achieves information-theoretic security and has $\mathcal{O}(N^{\frac{1}{3}})$ communication complexity. Since then, there has been a wide range of research works on two-server PIR [16], [17], [19], [28]. Among various works, the two-server PIR schemes built on distributed point functions (DPF) are specially impressive, as they achieve $\mathcal{O}(\lambda \log_2 N)$ communication complexity [28]. As a practical tool, DPF-based PIR has inspired numerous works in the domain of secure database search [18], [31]. However, DPF-based PIR schemes need to tolerate $\mathcal{O}(N)$ computational complexity on the server side, i.e., the server needs to linearly scan the N-entry database to respond PIR queries. The computation limitation is formalized and proved by Beimal et al. [20]. To bypass the barrier, they introduced a preprocessing model to realize sublinear server computational complexity but at the costs of increasing the communication overheads.

Offline/Online PIR with Preprocessing: Following Beimal et al. 's idea, numerous subsequent works have been proposed [9], [12], [13], [14], [21], [32], [33]. The idea of PIR with preprocessing can be summarized as follows: the unavoidable linear server computation is shifted into a preprocessing offline phase, where auxiliary databases or hints are created, and a client can make PIR queries and get responses in the subsequent online phase at sublinear computational costs in N, using the previously generated hints. The preprocessing model can be divided into two categories: 1) a global preprocessing model where servers prepare an encoding of the database for all clients during the offline preprocessing phase; and 2) a client-specific preprocessing model where individual clients work with the servers to create and store their own sets of hints during the offline preprocessing phase. Under the global preprocessing model, Canetti et al. [32] and Lin et al. [33] developed the well-known doubly efficient PIR schemes, which rely on heavy theoretical tools such as oblivious locally decodable codes and fully homomorphic encryption, endering them largely theoretical thus far. Compared to them, the PIR schemes under the client-specific preprocessing model are more practical for deployment. Corrigan-Gibbs and Kogan proposed the first amortized sublinear offline/online PIR scheme under the two-server settings, which have practical instantiations.

And they further extend their scheme to avoid parallel repetitions for each PIR queries [12], [13], with $\mathcal{O}(\sqrt{N})$ online server computational complexity. Two constructions are provided, one of which is based on the utilization of publicly puncturable PRFs, which can achieve $\mathcal{O}(\lambda \log_2 \sqrt{N})$ online uplink communication complexity, but does not support efficient membership testing. Another is based on pseudorandom permutation (PRP) with efficient membership testing, but can only achieve $\mathcal{O}(\sqrt{N} \cdot \log_2 N)$ online uplink communication complexity without the property of being puncturable. Although some works put good efforts on designing privately puncturable PRF, these works are difficult to be instantiated [34], [35]. Recently, Lazzaretti et al. proposed a technique named weak privately puncturable PRF under a weaker cryptographic assumption, which supports both efficient implementations and membership testing. Nevertheless, their scheme results in an online downlink communication complexity of $\mathcal{O}(\sqrt{N} \cdot \ell)$ instead of $\mathcal{O}(\ell)$. This becomes a performance bottleneck when the database record is large.

Different from the above-mentioned PIR schemes under the client-specific preprocessing model, which impose on clients the burden of storing hints correlated to the database record size, the proposed two-server offline/online PIR scheme offers a significant reduction in client storage demands. Notably, the storage requirement for clients is decoupled from the size of the database content, thereby enhancing storage efficiency without compromising the sublinear computational complexity on the server side. Besides aiming for improving efficiency, there is another branch of research. The offline/online PIR schemes, originally designed for the two-server model, have been adapted in recent works for a single-server setting. This is achieved either through the use of fully homomorphic encryption or by streamlining the offline phase [12], [21], [22]. Moreover, there exists some parallel research that aims to extend these schemes to support mutable databases. [13], [14].

VIII. Conclusion

We have proposed a two-server offline/online private information retrieval scheme, named PIRS. PIRS can leverage secret sharing and oblivious switching to ensure the security in an outsourcing environment, and significantly reduce client storage while maintaining sublinear online computational complexity. To facilitate fast hint locating without additional client storage or heavy computations, we have also developed a novel method that allows a client to delegate the computations of hint locating to database servers. In addition, we have implemented a proof-of-concept prototype to validate PIRS's practicality, and performance evaluation results highlight its potential as a foundational component for constructing secure applications. For future work, we will explore how to effectively apply PIRS to diverse applications.

References

[1] S. Hu, L. Y. Zhang, Q. Wang, Z. Qin, and C. Wang, "Towards private and scalable cross-media retrieval," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1354–1368, May/Jun. 2021.

[2] M. Zhang et al., "Privacy-preserved data trading via verifiable data disturbance," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 4, pp. 3126–3140, Jul./Aug. 2024.

[3] A. Vadapalli, K. Storrier, and R. Henry, "Sabre: Sender-anonymous messaging with fast audits," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 1953–1970.

[4] K. Zhong, Y. Ma, and S. Angel, "Ibex: Privacy-preserving ad conversion tracking and bidding," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 3223–3237.

[5] D. Kales, O. Omolola, and S. Ramacher, "Revisiting user privacy for certificate transparency," in *Proc. Eur. Symp. Secur. Privacy*, 2019, pp. 432–447.

[6] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish, "Scalable and private media consumption with popcorn," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2016, pp. 91–107.

[7] S. Angel, H. Chen, K. Laine, and S. Setty, "PIR with compressed queries and amortized query processing," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 962–979.

[8] S. J. Menon and D. J. Wu, "Spiral: Fast, high-rate single-server PIR via FHE composition," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 930–947.

[9] 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 *Proc. USENIX Secur.*, 2023, pp. 1–13.

[10] J. Liu, J. Li, D. Wu, and K. Ren, "PIRANA: Faster multi-query PIR via constant-weight codes," in *Proc. IEEE Symp. Secur. Privacy*, 2024, pp. 4315–4330.

[11] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, 1998.

[12] H. Corrigan-Gibbs and D. Kogan, "Private information retrieval with sublinear online time," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2020, pp. 44–75.

[13] D. Kogan and H. Corrigan-Gibbs, "Private blocklist lookups with checklist," in *Proc. USENIX Secur.*, M. Bailey and R. Greenstadt, Eds., 2021, pp. 875–892.

[14] Y. Ma, K. Zhong, T. Rabin, and S. Angel, "Incremental offline/online PIR," in *Proc. USENIX Secur.*, 2022, pp. 1741–1758.

[15] C. Huang, A. Yang, D. Liu, R. Lu, and X. Shen, "Two-server private information retrieval with high throughput and logarithmic bandwidth," in *Proc. IEEE/CIC Int. Conf. Commun. China*, 2023, pp. 1–6.

[16] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2014, pp. 640–658.

[17] S. M. Hafiz and R. Henry, "A bit more than a bit is more than a bit better," in *Proc. Privacy Enhancing Technol.*, 2019, vol. 4, pp. 112–131.

[18] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica, "Waldo: A private time-series database from function secret sharing," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 2450–2468.

[19] L. de Castro and A. Polychroniadou, "Lightweight, maliciously secure verifiable function secret sharing," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2022, pp. 150–179.

[20] A. Beimel, Y. Ishai, and T. Malkin, "Reducing the servers' computation in private information retrieval: PIR with preprocessing," in *Proc. Annu. Int. Cryptol. Conf.*, 2000, pp. 55–73.

[21] A. Lazzaretti and C. Papamanthou, "TreePIR: Sublinear-time and polylog-bandwidth private information retrieval from DDH," in *Proc. Annu. Int. Cryptol. Conf.*, 2023, pp. 284–314.

[22] M. Zhou, A. Park, E. Shi, and W. Zheng, "Piano: Extremely simple, single-server PIR with sublinear server computation," in *Proc. IEEE Symp. Secur. Privacy*, 2024, pp. 4296–4314.

[23] Y. Zheng, R. Lu, Y. Guan, J. Shao, and H. Zhu, "Efficient and privacy-preserving similarity range query over encrypted time series data," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 4, pp. 2501–2516, Jul./Aug. 2022.

[24] Y. Zheng, H. Duan, C. Wang, R. Wang, and S. Nepal, "Securely and efficiently outsourcing decision tree inference," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 1841–1855, May/Jun. 2022.

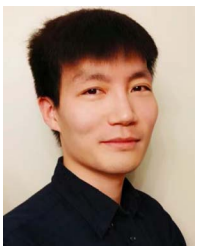
[25] Y. Du, H. Duan, L. Xu, H. Cui, C. Wang, and Q. Wang, "PEBA: Enhancing user privacy and coverage of safe browsing services," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 5, pp. 4343–4358, Sep./Oct. 2023.

[26] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 19–38.

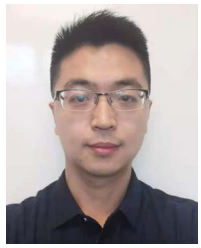
[27] L. Luo, D. Guo, R. T. Ma, O. Rottenstreich, and X. Luo, "Optimizing bloom filter: Challenges, solutions, and comparisons," *IEEE Commun. Surv. Tut.*, vol. 21, no. 2, pp. 1912–1949, Secondquarter 2019.

[28] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1292–1303.
[29] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Math. Cryptol.*, vol. 9, no. 3, pp. 169–203, 2015.
[30] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 830–842.
[31] C. Huang, D. Liu, A. Yang, R. Lu, and X. Shen, "Multi-client secure and efficient DPF-based keyword search for cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 1, pp. 353–371, Jan./Feb. 2024.
[32] R. Canetti, J. Holmgren, and S. Richelson, "Towards doubly efficient private information retrieval," in *Proc. Theory Cryptogr. Conf.*, 2017, pp. 694–726.
[33] W.-K. Lin, E. Mook, and D. Wichs, "Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE," in *Proc. Annu. ACM Symp. Theory Comput.*, 2023, pp. 595–608.
[34] D. Boneh, K. Lewi, and D. J. Wu, "Constraining pseudorandom functions privately," in *Proc. IACR Int. Workshop Public Key Cryptogr.*, 2017, pp. 494–524.
[35] E. Shi, W. Aqeel, B. Chandrasekaran, and B. Maggs, "Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time," in *Proc. Annu. Int. Cryptol. Conf.*, 2021, pp. 641–669.

Cheng Huang (Member, IEEE) received the PhD degree in electrical and computer engineering from the University of Waterloo, ON, Canada, in 2020. From 2020 to 2023, he was a postdoctoral research fellow with the Department of Electrical and Computer Engineering, University of Waterloo. He is currently an associate professor with the College of Computer Science and Artificial Intelligence, Fudan University, Shanghai, China. He has authored more than 80 peer-reviewed articles in premier journals and conferences, including *IEEE Transactions on Dependable and Secure Computing*, *IEEE Journal on Selected Areas in Communications*, and AAAI. His publications have received more than 2,500 citations, with an h-index of 29, according to Google Scholar. His research interests include data privacy, AI security, and trustworthy intelligent systems. Dr. Huang is an associate editor for several international journals, including *IEEE Transactions on Dependable and Secure Computing*, *IEEE Internet of Things Journal*, and Springer PPNA. He was the symposium co-chair of IEEE GLOBECOM 2024, guest editor of *Security and Safety (S&S)*, and publicity chair of ICA3PP 2022 and PST 2023. He was the recipient of five Best Paper Awards at international conferences, including IEEE ICC, GLOBECOM, and ICCC. He is the board governor representative of IEEE Blockchain Technical Committee.



Anjia Yang (Member, IEEE) received the PhD degree with the Department of Computer Science, City University of Hong Kong, in 2015. He was a postdoctoral with the City University of Hong Kong from 2015 to 2016 and also with Jinan University, Guangzhou from 2016 to 2019. From 2018 to 2019, he was a visiting scholar with BBCR group, University of Waterloo. He is currently a professor with Jinan University. He has authored or coauthored more than 60 international papers including journals and conferences, such as *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Intelligent Transportation System*, NDSS, ASIACRYPT, and ESORICS. His research interests include security and privacy in Internet of Things, vehicular networks, blockchain and cloud computing. He was the PC members or organizers of more than 30 international conferences. He is also the editors of journals such as *Security and Communication Networks*, *Symmetry*, and *Blockchain*.



Dongxiao Liu (Member, IEEE) received the PhD degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada in 2020. He was a postdoctoral research fellow with the Department of Electrical and Computer Engineering, University of Waterloo, from 2020 to 2023. He is currently a professor with the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interests include security and privacy in communication networks and blockchain.

Rongxing Lu (Fellow, IEEE) received the PhD degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2012. He was a postdoctoral fellow with the University of Waterloo from 2012 to 2013. From 2013 to 2016, he was an assistant professor with the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore. He is currently the Mastercard IoT research chair, university research scholar, and professor with the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. He has authored or coauthored extensively in his research interests which include applied cryptography, privacy enhancing technologies, and IoT-Big Data security, and privacy. He was the recipient of nine best (student) paper awards from some reputable journals and conferences. He was awarded the most prestigious "Governor General's Gold Medal," when he received his PhD degree. He has won the Eighth IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award in 2013. He is the chair of IEEE Communications and Information Security Technical Committee (ComSoc CIS- TC) and founding co-chair of IEEE TEMS Blockchain and Distributed Ledgers Technologies Technical Committee (BDLT-TC). He is the Winner of 2016-2017 Excellence in Teaching Award, FCS, and UNB.

Xuemin (Sherman) Shen (Fellow, IEEE) received the PhD degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is currently a university professor with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research interests include network resource management, wireless network security, Internet of Things, 5G and beyond, and vehicular networks. Dr. Shen is a registered professional engineer of Ontario, Canada, Engineering Institute of Canada fellow, Canadian Academy of Engineering fellow, Royal Society of Canada fellow, Chinese Academy of Engineering foreign member, and distinguished lecturer of the IEEE Vehicular Technology Society and Communications Society. Dr. Shen was the recipient of "West Lake Friendship Award" from Zhejiang Province in 2023, President's Excellence in Research from the University of Waterloo in 2022, Canadian Award for Telecommunications Research from the Canadian Society of Information Theory (CSIT) in 2021, R.A. Fessenden Award in 2019 from IEEE, Canada, Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, Joseph LoCicero Award in 2015 and Education Award in 2017 from the IEEE Communications Society (ComSoc), Technical Recognition Award from Wireless Communications Technical Committee (2019) and AHSN Technical Committee (2013), Excellent Graduate Supervision Award in 2006 from the University of Waterloo, and Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He is/was the general chair of 6 G Global Conference'23 and ACM Mobihoc'15, technical program committee chair/co-chair of IEEE Globecom'24, 16 and 07, IEEE Infocom'14, IEEE VTC'10 Fall, and chair of IEEE ComSoc Technical Committee on Wireless Communications. Dr. Shen is the president of IEEE ComSoc. He was the vice president of Technical & Educational Activities, vice president of Publications, member-at-large on the Board of Governors, chair of the Distinguished Lecturer Selection Committee, and member of IEEE Fellow Selection Committee of the ComSoc. Dr. Shen was the editor-in-chief of *IEEE Internet of Things Journal*, *IEEE Network*, and *IET Communications*.