Private Information Retrieval: A Tutorial and Survey

Pranav Shriram Arunachalaramanan, Yue Chen, and Ling Ren

***Abstract*—Private information retrieval (PIR) is a fundamental primitive for protecting user privacy. It enables a user to retrieve entries from a public database without revealing which entries are being retrieved. PIR has been studied in many settings, e.g., with information-theoretic or computational security, with a single server or multiple non-colluding servers, and with or without preprocessing to the database. In this article, we describe several PIR schemes that we believe are accessible to readers without prior knowledge in PIR. Although conceptually simple, these schemes capture the main ideas underlying mainstream design paradigms. We also describe extensions of PIR that support keyword queries and batch queries. Beyond describing the schemes themselves, we characterize the concrete efficiency of different PIR paradigms, provide guidance on selecting a paradigm in practice, and discuss practical applications of PIR. We hope this article helps readers understand the current research landscape in PIR and serves as a starting point for exploring more advanced topics in the field.**

***Index Terms*—Private information retrieval, PIR, privacy-preserving applications, homomorphic encryption, PIR with preprocessing, stateful PIR, keyword PIR, batch PIR**

I. Introduction

Can a user Alice read an article on a news website without exposing *which* article she is reading? In this case, the content of the article is public, but Alice's choice of article may reveal sensitive information about her personal interests, political leanings, health conditions, professional activities, and so on. Alice would therefore prefer that no one, neither the news website nor any observer in the network, learn which article she accessed. This example highlights a fundamental privacy challenge: can a user obtain public information without revealing what information one seeks?

Private Information Retrieval (PIR) [1] is a cryptographic primitive that solves this exact problem. PIR allows users (also called clients) to retrieve entries from a public database [1] hosted on the server without revealing any information about *which* entries are retrieved.

As one of the most fundamental cryptographic primitives for privacy protection, PIR has been deployed or suggested for a variety of applications besides the above example of reading news articles. Apple has deployed PIR in its Live Caller ID Lookup and URL filtering features [2]. PIR has also been suggested for applications such as anonymous communication [3]–[5], password breach alerting [6], [7], secure advertising [8]–[10], and privacy-preserving media streaming [11]. We describe several of these applications in more detail in Section XI.

All three authors are with the Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign, Urbana, IL 61801, USA. Emails: {psa3,yuec12,renling}@illinois.edu.

[1]The term *database* in the context of PIR simply refers to a set of generic data entries. It does *not* refer to organized information or its management system, such as relational or SQL databases.

PIR schemes have been studied in various models and can be classified based on the modeling choices. We consider three major aspects of the model.

- **Number of servers**. The client can interact with a single server or multiple non-colluding servers. Having multiple non-colluding servers is a strong trust assumption, but it can decrease the cost of PIR, sometimes dramatically.
- **Computational assumptions**. The PIR scheme can be either information-theoretically secure or computationally secure. A computationally secure PIR, as is customary in cryptography, relies on some cryptographic assumptions, i.e., the adversary is computationally bounded and unable to solve some computational problem. In some settings, cryptographic assumptions are necessary for PIR; in other settings, they can substantially reduce the cost of PIR.
- **Preprocessing**. Either the server(s) or the client can perform preprocessing on the database. Preprocessing the database can speed up subsequent PIR invocations; however, the preprocessing step itself can be expensive and may impose significant changes and constraints on the system design and deployment model.

Based on the combination of the above aspects of the model, different design paradigms for PIR have emerged. In this article, we present five conceptually simple schemes that reflect the key design principles and techniques in different paradigms. We also briefly review the literature in each setting. We hope that this article helps readers develop a basic understanding of mainstream PIR design paradigms and serves as a starting point for exploring more advanced PIR schemes.

We start with PIR *without* preprocessing, which is the more standard and better studied setting. (In this article, a PIR scheme refers to one without preprocessing by default. PIR schemes with preprocessing will be explicitly stated as such.) We describe three simple and representative PIR schemes:

- A two-server information-theoretic PIR scheme using just bitwise XORs (Section III);
- A two-server computational PIR scheme based on Distributed Point Functions (DPF) (Section IV); and
- A single-server computational PIR scheme based on homomorphic encryption (Section V).

For the last combination of a single server and information-theoretic security (without preprocessing), no interesting scheme exists as the trivial scheme of downloading the entire database is optimal [1].

All PIR schemes without preprocessing must incur server computation costs linear in the database size. Preprocessing the database by either the servers or the clients can circumvent the linear server computation bottleneck. Existing server-preprocessing PIR schemes with sublinear server computation

require advanced techniques and are beyond the scope of this article. Thus, we describe:

- A single-server server-preprocessing PIR scheme with linear but improved server computation (Section VI).
- A two-server client-preprocessing PIR scheme with sublinear server computation (Section VII).

In the original formulation of PIR (with or without preprocessing), the entries in the database are identified by consecutive integers. The schemes in Sections III to VII all fall into this category. However, in most practical applications, including the ones mentioned at the beginning of the article, the database entries are instead identified by keywords or arbitrary strings. This variant of PIR is called *keyword PIR*. Another useful variant of PIR is *batch PIR*, in which the client retrieves multiple entries from the server's database at a time. We discuss these two variants of PIR and describe simple schemes for them in Section VIII and Section IX.

We also briefly explain PIR's connections to other cryptographic primitives in Section II-D, characterize the practical performance of different paradigms of PIR, and provide guidelines for selecting an appropriate paradigm in practice in Section X.

II. Private Information Retrieval

A. Problem Definition and Interface

Private Information Retrieval (PIR) is a protocol between a client and one or more servers. It allows the client to retrieve an entry from a public database hosted on the server(s), without revealing to the server(s) *which* entry is retrieved.

The database DB consists of N entries. Theoretical explorations of PIR often focus on the simplest case where each entry is a single bit. In this article, we consider the more general and practical setting with an arbitrary entry size of w bits. In most of this article, we assume the N entries are indexed by integers from 1 to N. Section VIII will deal with a database whose entries are indexed by arbitrary keywords instead of consecutive integers.

The database is public. Many clients, potentially anyone with an Internet connection to the server(s), can interact with the server(s) to retrieve their respective desired entries from the same database. Each server stores a single copy of the database (possibly preprocessed) regardless of the number of clients it interacts with over time.

The interaction between the client and the server in a single-server PIR scheme is shown in Figure 1. The client has an entry of interest that it wishes to retrieve. Let x denote the index of the client's desired entry, also called the desired index for short. Given the desired index x, the client constructs a query q and sends q to the server. From the client's query q and the database DB, the server computes a response r and sends r to the client. Finally, using the server response r, the client computes an answer, which should be equal to the desired entry $\mathsf{DB}[x]$.

In a multi-server PIR scheme, the client's query consists of multiple components, one component per server. Each server computes a response and sends it to the client. The client computes the answer using all the responses.

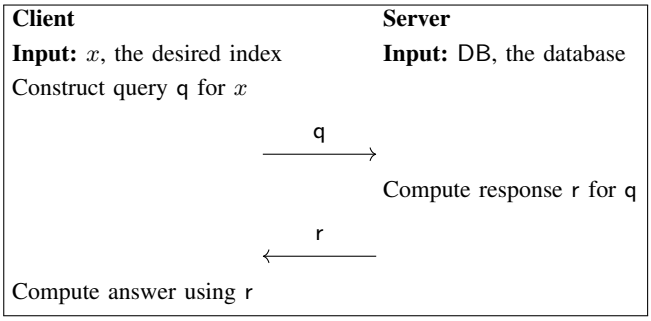


Fig. 1. The interaction between the client and the server in a single-server PIR protocol.

TABLE I
Common Notations

Symbol	Meaning
$[a]$	The set $\{1, 2, \ldots, a\}$
DB	The database
N	Number of entries in DB
w	Entry size in bits
x	Index or keyword of the client's desired entry
q	Client's query
r	Server's response
$\oplus$	Bitwise XOR
$\langle \cdot, \cdot \rangle$	Inner product

Common notations used in this article are summarized in Table I.

B. Correctness and Privacy

A PIR scheme must satisfy correctness and privacy, defined as follows:

- **Correctness:** The answer computed by the client in the end equals the desired entry.
- **Privacy:** Informally, the server learns no information about the client's desired index from the client's query.

In this article, we assume the server(s) are *honest but curious*, i.e., the server(s) faithfully execute the steps of the PIR scheme but may attempt to infer a client's desired index by analyzing the client's query. In the single-server setting, the server sees everything that an adversary in the network sees. In the multi-server setting, it is further assumed that the servers do not collude with one another. There is also an implicit assumption that no adversary in the network can see the client's query to more than one server. Thus, in either setting, we can treat the server(s) as the only adversary.

We also note that the PIR problem is not concerned with the server's privacy. In particular, the client is allowed to learn information about the *other entries* in the database. This is again because the entire database on the server is assumed to be public. There are cryptographic primitives that do worry about the server's privacy. We will briefly contrast them with PIR in Section II-D.

We now make the definition of privacy more formal. Let X be the random variable representing the client's desired index. X follows some arbitrary distribution. Let Q_i be the random variable representing the client's query to the i-th server. For an *information-theoretic* PIR scheme, the distribution of X

conditioned on any server's observed query must be the same as the a priori distribution of X. That is, for each server i and every index $x \in [N]$,

$$\Pr[X = x \mid Q_i = q_i] = \Pr[X = x].$$

This captures the intuition that the client's query to each server does not leak any information about the client's desired index.

For computational PIR, it is more convenient to think of PIR privacy as a game between the client and the server(s), analogous to the indistinguishability game in encryption. The game is defined as follows for single-server PIR.

1) The server selects two indices x_0 and x_1 and sends them to the client.
2) The client flips a random coin b and executes the PIR protocol using x_b as the desired index.
3) Based on the client's query, the server tries to guess b.

Intuitively, if the server cannot guess b with better accuracy than a random guess, then the server did not learn any information from the client's query, and we say the client wins the game. More formally, if no computationally bounded server can correctly guess b with a probability better than $0.5 + \epsilon$ where ϵ is negligible (in some security parameter), then the PIR scheme satisfies computational privacy.

In the case of multi-server schemes, each server plays the game with the client and tries to guess b based on the client's query to it. The scheme satisfies privacy if the client wins the indistinguishability game with each server.

C. Efficiency Metrics

The main efficiency metrics for PIR (without preprocessing) are the communication and server computation costs.

- **Communication**: the amount of data (typically measured in bits) exchanged between the client and the server(s). Since there is a large gap between upload and download speeds in modern Internet, it is helpful to break down the communication cost into *query size* and *response size*.
- **Server computation**: the amount of computation performed by the server(s) to compute the response.

Although the client also performs some computation to construct the query and compute the answer, it is usually much less than the server computation and hence not a focus in PIR schemes without preprocessing. When we later move to PIR schemes with preprocessing, client computation and storage costs (of either clients or servers) become additional important efficiency metrics.

A trivial scheme. A trivial PIR scheme is for the server to send the entire database to the client. It is easy to see that the trivial scheme achieves correctness and privacy, and has $O(Nw)$ communication cost (response size to be more precise). We are interested in *nontrivial* PIR schemes that have communication sublinear in N.

D. Connections to Other Cryptographic Primitives

We briefly summarize the connections of PIR to other fundamental cryptographic primitives. Readers unfamiliar with these cryptographic primitives can skip this subsection and proceed with the rest of the article without difficulty.

PIR vs. computational security and one-way functions. Multi-server PIR schemes with information-theoretic privacy and sublinear in N communication exist, as we will soon see in Section III. However, a nontrivial single-server PIR scheme, i.e., one whose communication cost is less than the database size, can only achieve computational privacy [1].

A nontrivial single-server PIR scheme implies one-way functions [12], oblivious transfer (OT) [13], and collision-resistant hash functions [14]. Because OT and collision-resistant hash functions are believed to be stronger primitives than one-way functions[2], it is unlikely to construct a nontrivial single-server PIR scheme from one-way functions alone. Indeed, the mainstream approach to single-server PIR, including the scheme described in Section V of this article, relies on homomorphic encryption.

PIR vs. Oblivious Transfer. PIR and OT [17] share some similarities, but also have crucial differences. It may be helpful to contrast the two. A 1-out-of-N OT is a protocol between two parties: a sender holding N entries $x_1, x_2, x_3, \ldots, x_N$ and a receiver with an index $i \in [N]$. At the end of the protocol, the receiver obtains x_i without revealing any information about i to the sender and without learning anything about the other entries held by the sender.

The OT sender resembles the server in PIR, and the OT receiver resembles the PIR client. Compared with PIR, OT additionally requires sender (server) privacy: the receiver is not allowed to learn anything about the sender's other entries. As mentioned, any nontrivial single-server PIR scheme can be transformed into an OT scheme [13]. But most PIR schemes in their original forms, including the trivial scheme of downloading the whole database, are not solutions to OT because they do not provide server (sender) privacy. This also means OT with $O(N)$ communication can still be interesting, but PIR with $O(N)$ communication is usually not.

PIR vs. Private Set Intersection. Private set intersection (PSI) [18], [19] is a protocol between the client and the server, where the client has a set of elements $C = \{c_1, c_2, \ldots, c_M\}$ and the server has a set of elements $S = \{s_1, s_2, \ldots, s_N\}$. At the end of the protocol, the client retrieves the elements in the intersection $C \cap S$ without learning any additional information about S, and the server learns no information about the elements in C.

Notice that if we let the client perform a batch keyword PIR (see Section IX) with C as its batch and S as the server's database, the client obtains the intersection $C \cap S$ without revealing any information to the server. However, PSI additionally requires server privacy. Additional techniques are required to transform a batch keyword PIR into PSI [20].

PIR vs. Oblivious RAM. Oblivious RAM (ORAM) [21] is the other main cryptographic primitive that hides which entry is accessed. An ORAM scheme allows a client to *outsource* its private database to a remote server. The client can then

[2]In fact, they provably cannot be constructed from one-way functions in a black-box manner [15], [16].

access any database entry without revealing to the server any information about which entry is accessed.

The key difference between PIR and ORAM is that PIR considers a public database, whereas ORAM deals with a client's private database. The database in PIR is public in the sense that many clients, possibly anyone on the Internet, can retrieve entries from it. The server stores a single copy of the database regardless of how many clients it interacts with. In contrast, in ORAM, a client's database is encrypted by that client and is not accessible to any other client. If an ORAM server supports multiple clients, it must allocate a separate storage space for each client. In return, ORAM can easily achieve amortized polylogarithmic complexity in N for both communication and server computation with small constants [22], whereas this remains a major open problem for PIR.

E. A Basic Recipe

A very simple structure underlies the three PIR schemes in Sections III to VI in this article. Let DB be a database with N entries, indexed by integers in $[N] = \{1, 2, 3, \ldots, N\}$. Given a desired index $x \in [N]$, let q be the one-hot encoding of x, that is, for all $i \in [N]$,

$$\mathsf{q}[i] = \begin{cases} 0, & i \neq x \\ 1, & i = x \end{cases} \tag{1}$$

Then, $\mathsf{DB}[x]$ can be written as an inner product between q and the database DB, defined as follows:

$$\mathsf{DB}[x] = \langle \mathsf{q}, \mathsf{DB} \rangle = \sum_{i=1}^{N} \mathsf{q}[i] \cdot \mathsf{DB}[i]$$

where $\cdot$ represents multiplication (by either 0 or 1) and the summation in $\sum$ can be either addition or bitwise XOR. Looking ahead, the inner products use bitwise XORs in Sections III and IV, and use additions in Sections V and VI.

Obviously, the client cannot send the one-hot vector q to the server(s) due to the privacy requirement of PIR. From here, a natural idea is to send q in a form that is "protected" by cryptography and still permits the above inner product operations. Common cryptographic tools for this purpose include *additive secret sharing* (when there are multiple non-colluding servers) and *homomorphic encryption* (when there is a single server). This basic recipe, combined with the appropriate cryptographic tools, leads to the schemes in Sections III to VI.

III. Two-server Information-theoretic PIR

In this section, we describe the first and simplest PIR schemes from the work of Chor et al. [1]. The schemes are information-theoretic and require two non-colluding servers.

A. A Strawman Scheme

We start with a simple strawman scheme. The two servers, denoted by S_1 and S_2, each hold a copy of the database DB. The client wishes to retrieve the x-th entry of DB. If preferred, the reader can think of each entry of DB to be a single bit for simplicity, although the scheme as presented will support an arbitrary entry size.

Recall the basic recipe in Section II-E. In this strawman scheme, the client sends the one-hot vector q to the two servers in a *secret-shared* form. The client picks a uniformly random binary vector q_1 of length N. Every coordinate in q_1 is either 0 or 1 with a half-half probability. Then, the client constructs q_2 as follows:

$$\mathsf{q}_2[i] = \begin{cases} \mathsf{q}_1[i], & i \neq x \\ 1 - \mathsf{q}_1[i], & i = x \end{cases}$$

In other words, q_2 is identical to q_1 except that the bit at the desired index x is flipped. Observe that q_1 and q_2 are *XOR shares* of the one-hot vector q, i.e., the bitwise XOR of q_1 and q_2 gives q:

$$\mathsf{q}_1 \oplus \mathsf{q}_2 = \mathsf{q}.$$

The client sends q_1 to server S_1 and q_2 to server S_2. Each server computes its response by taking the inner product between the query and the database. That is,

$$\mathsf{r}_1 = \langle \mathsf{q}_1, \mathsf{DB} \rangle = \bigoplus_{i=1}^{N} \mathsf{q}_1[i] \cdot \mathsf{DB}[i],$$
$$\mathsf{r}_2 = \langle \mathsf{q}_2, \mathsf{DB} \rangle = \bigoplus_{i=1}^{N} \mathsf{q}_2[i] \cdot \mathsf{DB}[i].$$

Finally, the client computes the answer by XORing r_1 and r_2. The pseudocode is given in Algorithm 1 and an illustration is shown in Figure 2.

Correctness. Observe that for all $i \neq x$, $\mathsf{q}_1[i] = \mathsf{q}_2[i]$. Thus, when the client computes $\mathsf{r}_1 \oplus \mathsf{r}_2$, all those database entries cancel out. The only term that remains is exactly $\mathsf{DB}[x]$ since $\mathsf{q}_1[x] \oplus \mathsf{q}_2[x] = 1$. In more detail,

$$\begin{aligned} \mathsf{r}_1 \oplus \mathsf{r}_2 &= \bigoplus_{i=1}^{N} (\mathsf{q}_1[i] \cdot \mathsf{DB}[i]) \oplus \bigoplus_{i=1}^{N} (\mathsf{q}_2[i] \cdot \mathsf{DB}[i]) \\ &= \bigoplus_{i=1}^{N} (\mathsf{q}_1[i] \cdot \mathsf{DB}[i]) \oplus (\mathsf{q}_2[i] \cdot \mathsf{DB}[i]) && (2) \\ &= \bigoplus_{i=1}^{N} (\mathsf{q}_1[i] \oplus \mathsf{q}_2[i]) \cdot \mathsf{DB}[i] && (3) \\ &= \bigoplus_{i=1}^{N} \mathsf{q}[i] \cdot \mathsf{DB}[i] && (4) \\ &= \mathsf{DB}[x] \end{aligned}$$

Because XOR is commutative and associative, we get Equation (2) by rearranging terms. We get Equation (3) because AND (which is equivalent to multiplication by a single bit) distributes over XOR. We get Equation (4) since q_1 and q_2 are XOR shares of q.

Privacy. We show the scheme satisfies the privacy definition in Section II. Let X be the distribution of the client's desired index. Let Q_1 and Q_2 be the random variables representing the client's queries to the two servers, respectively. Each bit in the

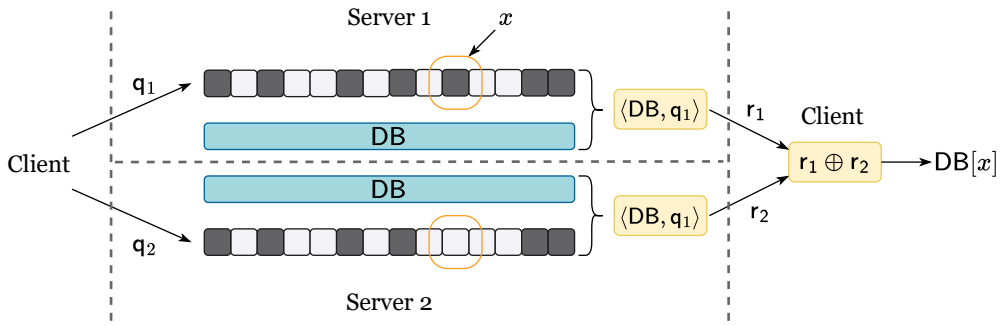
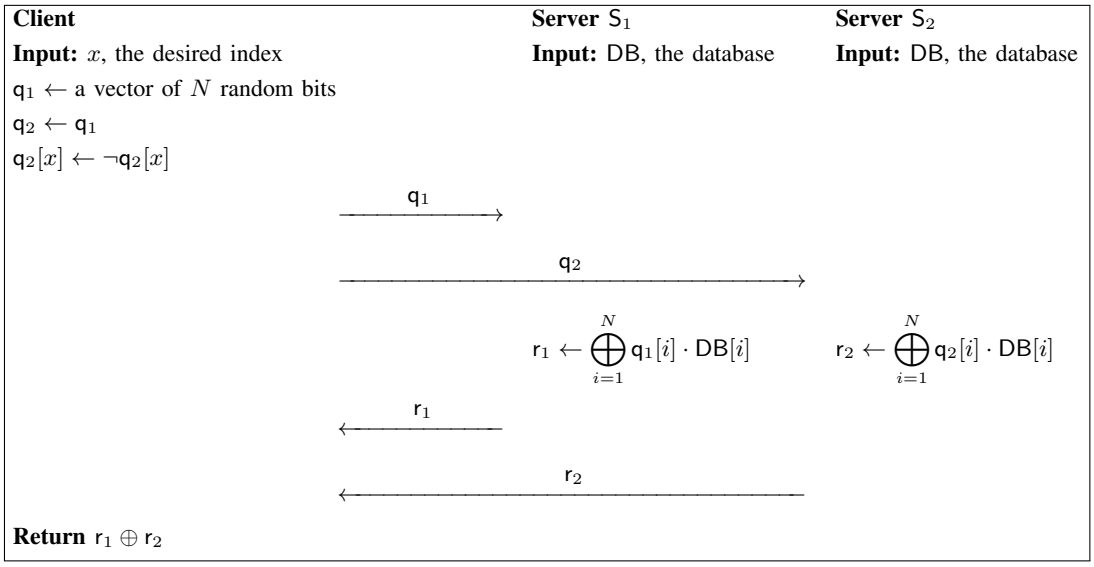
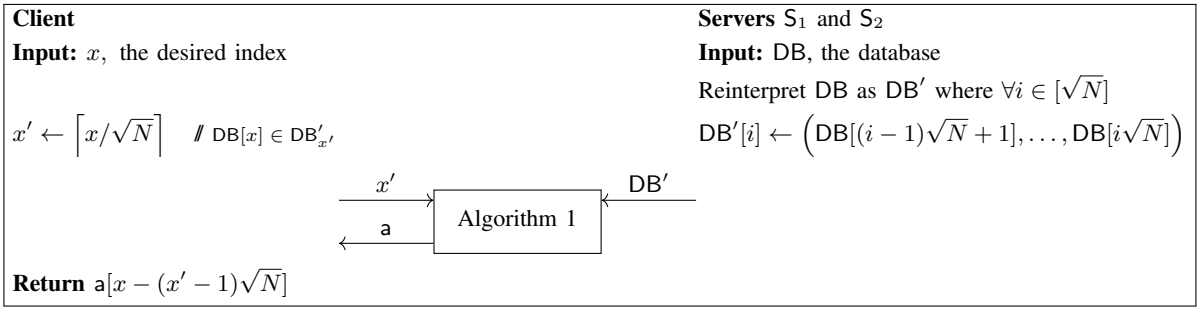


Fig. 2. An illustration of the strawman two-server information-theoretic PIR scheme.



Algorithm 1. A strawman two-server information-theoretic PIR scheme.



Algorithm 2. Two-server information-theoretic PIR scheme with $O(\sqrt{N})$ communication.

vector Q_1 is sampled uniformly at random and independent of the client's desired index. Hence, we have

$$\Pr[X = x \mid Q_1 = \mathsf{q}_1] = \Pr[X = x].$$

Q_2 is identical to Q_1 except that its x-th bit is flipped. When viewed independently of Q_1, Q_2 is also uniformly random and independent of the client's desired index, i.e.,

$$\Pr[X = x \mid Q_2 = \mathsf{q}_2] = \Pr[X = x].$$

Thus, the scheme satisfies privacy.

Efficiency. In this scheme, the client sends N bits to each server and receives a w-bit response from each server. The communication cost of the scheme is hence $O(N + w)$. The server computation cost is $O(Nw)$.

We call this simple scheme a strawman because of its $O(N)$ query size. Indeed, if the entries are small, e.g., $w = O(1)$, the scheme is no better than the trivial scheme of downloading the whole database. But if the entries are very large relative to N, e.g., $w = \Omega(N)$, then this "strawman" scheme can be competitive or even close to optimal, since fetching the entry of interest alone (without privacy) needs $O(w)$ communication and the $O(N)$ query size is no longer the bottleneck.

B. Reducing the Query Size

Building on the strawman scheme in Algorithm 1, Algorithm 2 is a two-server information-theoretic PIR scheme with $O(\sqrt{N}w)$ communication cost for a database DB with N entries. The idea is to reinterpret the database as having fewer but larger entries to balance the sizes of the query and the response.

Both the client and the two servers can reinterpret DB as a new database DB' with $\sqrt{N}$ larger entries. The first $\sqrt{N}$ entries of DB become $\mathsf{DB}'[1]$, the next $\sqrt{N}$ entries become $\mathsf{DB}'[2]$, and so on. The x-th entry in the original database DB is now part of the $\left\lceil \frac{x}{\sqrt{N}} \right\rceil$-th entry of DB'. The client and the servers execute the strawman scheme in Algorithm 1 using $x' = \left\lceil \frac{x}{\sqrt{N}} \right\rceil$ and DB'. At the end of the strawman scheme, the client has $\mathsf{DB}'[x']$ and easily recovers $\mathsf{DB}[x]$.

The correctness and privacy of this scheme are straightforward from the correctness and privacy of the strawman scheme in Algorithm 1.

Efficiency. Each server performs $O(N)$ w-bit XOR operations, so the server computation cost remains $O(Nw)$. The client sends $\sqrt{N}$ bits to each server and receives $\sqrt{N}w$ bits from each server. Hence, the communication cost of Algorithm 2 is $O(\sqrt{N}w)$.

The communication cost can be further improved to $O(\sqrt{Nw})$ by balancing the query and response sizes. The only modification needed is to instead interpret DB' as a database with $O(\sqrt{Nw})$ entries each of $O(\sqrt{Nw})$ bits.

C. A Brief Survey of Multi-server Information-theoretic PIR

The line of work in this setting generally considers databases with single-bit entries and focuses on minimizing the query size. In the paper by Chor et al. that introduced PIR [1], the authors gave various schemes in the multi-server information-theoretic setting. The two-server $O(\sqrt{N})$ scheme in Algorithm 2 is one of them. They also give a two-server scheme with $O(\sqrt[3]{N})$ communication cost by leveraging covering codes, as well as various schemes that use more than two servers.

In the two-server setting, the PIR scheme of Dvir and Gopi [23] achieves $N^{O(\sqrt{\log\log N/\log N})}$ communication. In the three-server setting, several works [24]–[26] achieve $2^{O(\sqrt{\log N \cdot \log\log N})}$ communication. In the k-server setting, the PIR scheme of Beimel et al. [27] achieves $N^{O(\frac{\log\log k}{k\log k})}$ communication.

IV. Two-server Computational PIR

In this section, we describe the two-server computational PIR scheme from Gilboa and Ishai [28]. The scheme leverages a cryptographic primitive called distributed point functions (DPF). Compared to the scheme in Section III, the DPF-based PIR scheme weakens the security from information-theoretic to computational; in return, it significantly improves the communication cost to depend logarithmically on N.

A. Point Function and Distributed Point Function

A *point function* is a function that outputs a non-zero value on one specific input, and outputs 0 on all other inputs. For the use case of PIR, we can assume, for simplicity, that the non-zero value is 1 and the domain is $[N]$. Formally, a point function with a parameter $\alpha \in [N]$ is $f_\alpha : [N] \to \{0,1\}$ where

$$f_\alpha(i) = \begin{cases} 1, & i = \alpha \\ 0, & i \neq \alpha \end{cases}$$

A *Distributed Point Function* (DPF), introduced by Gilboa and Ishai [28], can be thought of as a way to secret-share a point function f_α between two parties so that the parameter α remains hidden. Concretely, a DPF is a pair of algorithms:

- $\mathsf{Gen}(\alpha, N)$ returns a pair of keys k_1 and k_2,
- $\mathsf{Eval}(k, i)$ returns $y \in \{0,1\}$.

A DPF must satisfy the following correctness and privacy properties.

- **Correctness.** With $(k_1, k_2) \leftarrow \mathsf{Gen}(\alpha, N)$, $\mathsf{Eval}(k_1, i)$ and $\mathsf{Eval}(k_2, i)$ are XOR secret shares of $f_\alpha(i)$, i.e., for all $i \in [N]$,
$$\mathsf{Eval}(k_1, i) \oplus \mathsf{Eval}(k_2, i) = f_\alpha(i).$$
- **Privacy.** An individual key, k_1 or k_2 alone, leaks no information about the point function parameter α.

At a high level, a DPF can be thought of as a more succinct way to secret-share a one-hot vector than the basic XOR secret sharing used in Section III. As such, a primary efficiency metric for DPF is the key size. The state-of-the-art DPF construction by Boyle et al. [29] has key size $O(\lambda \log N)$ where λ is a security parameter. We will not delve into the details of the DPF construction since the PIR scheme (also proposed in [28]) uses DPF in a black-box manner.

B. The PIR Scheme

The setting is the same as in Section III. The database DB consists of N entries, indexed by integers in $[N]$. The two servers S_1 and S_2 each hold a copy of DB. The client's desired index is x and wants to retrieve $\mathsf{DB}[x]$.

The high-level idea of this scheme is also similar to that in Section III: the two servers will receive XOR secret shares of the one-hot vector q. But this time, we will rely on DPFs. In particular, observe that q can be derived from a point function: for each $i \in [N]$, $\mathsf{q}[i] = f_x(i)$.

The PIR scheme then follows naturally. The client computes $(k_1, k_2) \leftarrow \mathsf{Gen}(x, N)$ and sends k_1 to S_1 and k_2 to S_2. Using k_1, the server S_1 computes a vector q_1: for all $i \in [N]$,

$$\mathsf{q}_1[i] = \mathsf{Eval}(k_1, i).$$

Similarly, server S_2 computes q_2 using k_2: for all $i \in [N]$,

$$\mathsf{q}_2[i] = \mathsf{Eval}(k_2, i).$$

The correctness of the DPF ensures that $\mathsf{q}_1 \oplus \mathsf{q}_2 = \mathsf{q}$. The privacy of DPF ensures that each server, receiving a single key, learns no information about the desired index x.

The rest of the PIR protocol is identical to the strawman scheme in Section III. Each server S_ℓ computes its response

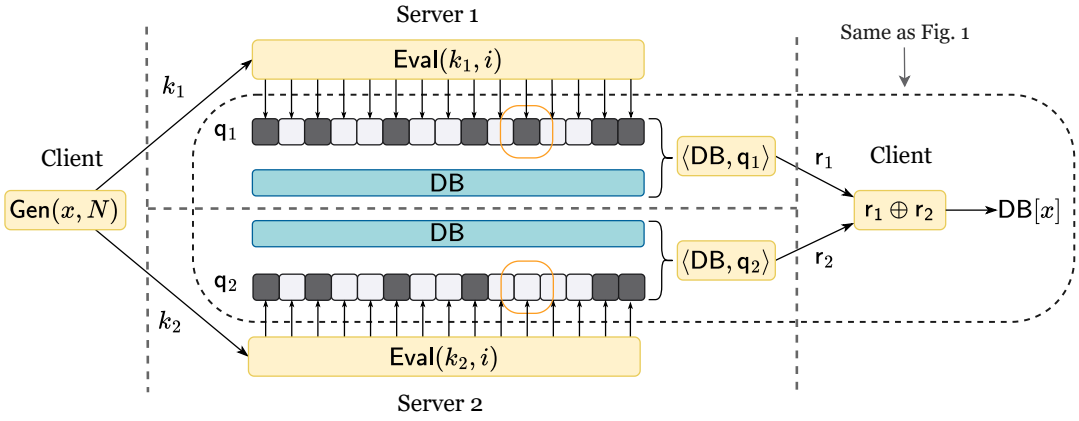
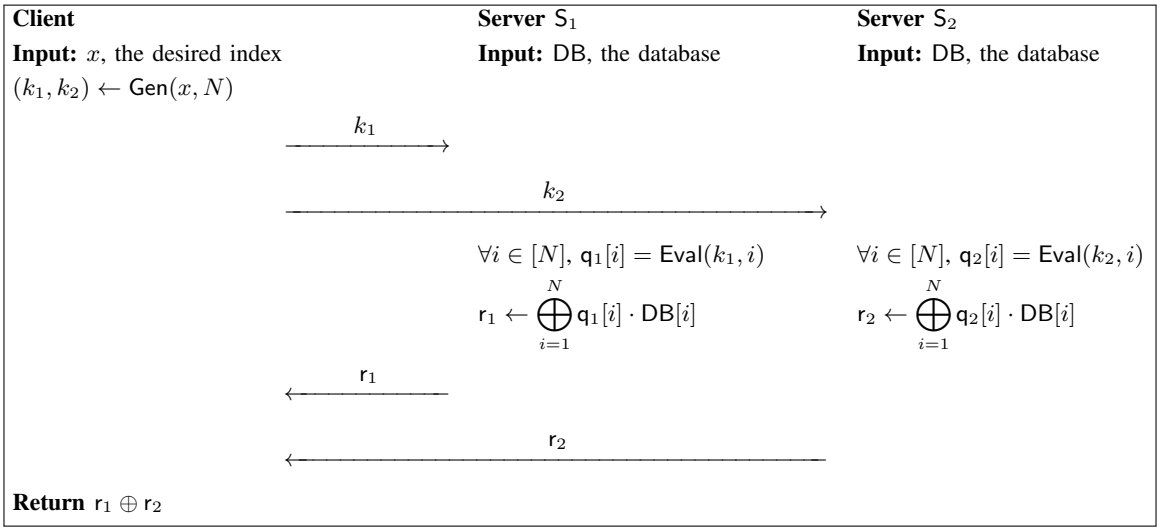


Fig. 3. An illustration of the DPF-based two-server computational PIR scheme.



Algorithm 3. Two-server computational PIR based on DPF.

$r_\ell = \bigoplus_{i=1}^{N} q_\ell[i] \cdot \mathsf{DB}[i]$. The client computes $r_1 \oplus r_2$ as the answer. The pseudocode of the scheme is given in Algorithm 3.

Efficiency. Using the state-of-the-art DPF [29], the two keys k_1 and k_2 are each $O(\lambda \log N)$ bits where λ is a security parameter. Thus, the query size is $O(\lambda \log N)$, which is much smaller than the information-theoretic schemes presented and surveyed in Section III. The response is a single entry of w bits from each server. The server computation of [29] involves $O(N/\lambda)$ calls to a pseudorandom function (PRF) and $O(N)$ w-bit XORs. The former comes from calling the DPF Eval function on all indices simultaneously (using additional optimizations introduced in [29]) and the latter comes from computing the inner product.

V. Single-server Computational PIR

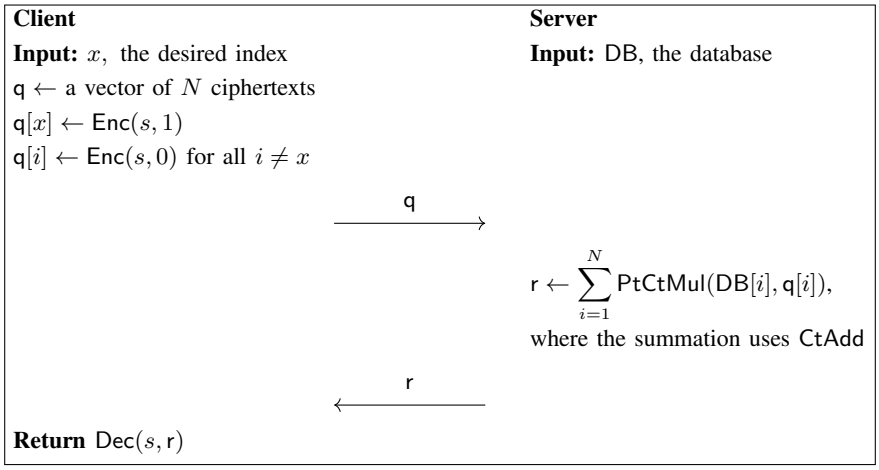
While having multiple non-colluding servers makes the PIR problem easier and enables more efficient solutions, it is a strong trust assumption and requires coordination from multiple organizations. Single-server PIR, though less efficient, is easier to deploy and has seen more adoption in practice [2].

As discussed in Section II-D, single-server PIR likely requires cryptographic assumptions and primitives beyond one-way functions. State-of-the-art single-server PIR schemes [7], [30]–[33] leverage Somewhat Homomorphic Encryption (SHE). We describe below the interface of SHE and a single-server PIR scheme that uses SHE in a black-box manner.

A. Somewhat Homomorphic Encryption

A Somewhat Homomorphic Encryption (SHE) is an encryption scheme that supports a limited number of addition and multiplication operations on ciphertexts. Enc takes a plaintext m and outputs a ciphertext encrypting m. Dec takes a ciphertext and gives back the plaintext. Both the Enc and Dec functions take a secret key s. We use the tilde notation $\widetilde{m}$ to denote an encryption of m.

CtAdd and CtMul are homomorphic operations on ciphertexts. They take two ciphertexts and output a ciphertext that encrypts the sum and product of the two underlying plaintexts, respectively. Lastly, PtCtMul takes a plaintext m_1 and an encryption of m_2 as input, and outputs a ciphertext that encrypts $m_1 \cdot m_2$. We list PtCtMul separately because it is



Algorithm 4. A strawman single-server computational PIR scheme. s is the client's secret key.

usually cheaper than CtMul. The interface of SHE is listed below.

- $\mathsf{Enc}(s, m) \to \widetilde{m}$
- $\mathsf{Dec}(s, \widetilde{m}) \to m$
- $\mathsf{CtAdd}(\widetilde{m_1}, \widetilde{m_2}) \to \widetilde{m_1 + m_2}$
- $\mathsf{CtMul}(\widetilde{m_1}, \widetilde{m_2}) \to \widetilde{m_1 \cdot m_2}$
- $\mathsf{PtCtMul}(m_1, \widetilde{m_2}) \to \widetilde{m_1 \cdot m_2}$

The state-of-the-art constructions of SHE are based on lattice cryptography [34], [35]. Examples include BFV [36], [37], BGV [38], and CKKS [39], all of which are based on the Ring Learning with Errors (RLWE) problem. In lattice cryptography, each ciphertext carries some *noise*, and the homomorphic operations (CtAdd, CtMul, and PtCtMul) increase the noise in the output ciphertext. When the noise exceeds a certain level, the decryption procedure Dec can no longer recover the plaintext. This is why SHE can only support a limited number of homomorphic operations. In state-of-the-art SHE, homomorphic additions (CtAdd) increase the noise slightly, while homomorphic multiplications (CtMul and PtCtMul) increase the noise substantially. Thus, SHE typically delivers good practical efficiency only on computation tasks with small multiplicative depth.

For the readers' reference, a cryptographic primitive closely related to SHE is *Fully Homomorphic Encryption* (FHE) [40]. FHE is a stronger primitive than SHE as it supports unlimited homomorphic additions and multiplications. Existing FHE constructions are also based on lattice cryptography and use noisy ciphertexts. In fact, many of the aforementioned RLWE-based schemes can operate as FHE schemes. In order to support unlimited homomorphic operations, an FHE scheme must periodically run a procedure called *bootstrapping*, which "refreshes" the ciphertexts to reduce the noise in them back to an acceptable level. The bootstrapping procedure is powerful but requires extremely expensive computation, making FHE orders of magnitude slower than SHE. As a result, practical single-server PIR schemes opt for SHE and need to find ways to control the multiplicative depth or noise growth.

B. A Strawman Scheme

We start with a strawman single-server PIR scheme. It builds upon the basic recipe in Section II-E and is conceptually similar to the strawman two-server scheme in Section III. The main idea is to send the one-hot query vector q after SHE encryption.

The encrypted one-hot vector hides the position of $\widetilde{1}$ and the desired index x, as a SHE scheme ensures that $\widetilde{0}$ is indistinguishable from $\widetilde{1}$. It is important to note that all modern encryption schemes, SHE schemes included, are *randomized*. Therefore, even though the encrypted vector contains many instances of $\widetilde{0}$, they all appear different and random, and are indistinguishable from the single $\widetilde{1}$ to the server.

The server computes the inner product between the database and the encrypted vector using homomorphic multiplications and additions, i.e., PtCtMul and CtAdd. The encrypted inner product result is sent back to the client as the response. The client decrypts and recovers the desired entry. The pseudocode is presented in Algorithm 4.

Handling arbitrary database entry sizes. The use of SHE raises a new issue worth clarifying: a SHE scheme has a fixed plaintext space, but databases come in various entry sizes. Indeed, we have made a simplifying assumption in Algorithm 4 that each database entry fits in the SHE plaintext space. If each database entry is longer than one plaintext, the PtCtMul function would need to split each entry into multiple chunks, just like what one would do when encrypting a long message. Each chunk fits in the plaintext space and multiplies with the ciphertext separately. The output also consists of multiple ciphertexts, each encrypting a portion of the result.

Equivalently, we can think of the database as multiple (say k) smaller databases with plaintext-sized entries. Now, fetching $\mathsf{DB}[x]$ is equivalent to retrieving the x-th entry from each of these smaller databases. Crucially, the desired index x is the same for all k smaller databases. Thus, the client only needs to send one query. The server can reuse the query for all k smaller databases. The server sends all k responses back to the client. The client assembles the desired entry $\mathsf{DB}[x]$ from the k responses. Note that this transformation works for all

PIR schemes. For example, we could interpret the two-server schemes in Algorithms 1 and 3 as applying this transformation to extend a base scheme for 1-bit entries to a scheme for longer entries.

Efficiency. The client sends N ciphertexts to the server. Assuming each entry fits in the plaintext space for simplicity, the server performs N PtCtMul and CtAdd operations and sends one ciphertext back to the client. Because a ciphertext is always larger than the underlying plaintext, uploading N ciphertexts is actually worse than downloading the entire database of N plaintext entries.

If each entry is as large as k plaintexts, then the server computation and the response size both increase by a factor of k, but the query size remains N ciphertexts. The total communication cost becomes $N + k$ ciphertexts, which can be significantly better than downloading the entire database of Nk plaintext entries, and even close to optimal if $k = \Theta(N)$. The server computation cost is $O(Nk)$ homomorphic operations, proportional to the database size.

C. Reducing the Query Size with Recursion

To improve communication, Algorithm 5 reduces the query size to $O(\sqrt{N})$ by recursively executing the strawman scheme twice, each time on a database with $\sqrt{N}$ entries.

To elaborate, the database is reinterpreted as having $\sqrt{N}$ segments, each containing $\sqrt{N}$ consecutive entries of DB. The client sends two encrypted one-hot vectors of size $O(\sqrt{N})$, corresponding to the selection of the desired segment, and the desired entry within that segment, respectively.

The server uses the PtCtMul and CtAdd operations to perform the inner product homomorphically in the first dimension. Here, each segment is treated as the plaintext input to PtCtMul. Note that a segment may be much larger than an SHE plaintext. As mentioned in Section V-B, internally, PtCtMul can split a segment into $\sqrt{N}$ parts and perform multiplication separately on each part.

The output of the first inner product is the encryption of the segment that contains the desired entry. It is treated as the intermediate database for the second dimension. For the second dimension, since the input database is encrypted, the server uses CtMul and CtAdd operations to perform the inner product homomorphically.

The correctness and privacy of the two-dimensional scheme in Algorithm 5 are straightforward as before. The query size is now $O(\sqrt{N})$ and the response is a single encrypted entry. Assuming for simplicity that each entry fits in the SHE plaintext space, the server computation still involves $O(N)$ homomorphic operations, as the first dimension still uses $O(N)$ PtCtMul and CtAdd operations.

Extension to d dimensions. The above two-dimensional scheme can be extended to d dimensions to further reduce the query size. The original database is viewed as a d-dimensional hypercube, where the length in each dimension is $\sqrt[d]{N}$. The PIR scheme can be thought of as a recursive one. The client's query consists of d encrypted one-hot vectors, each of size $O(\sqrt[d]{N})$. At each recursion level, the server homomorphically performs inner products along one dimension, "flattening" the hypercube along that dimension. A three-dimensional scheme is illustrated in Figure 4.

Increasing the number of recursion levels d reduces the communication cost to $O(d\sqrt[d]{N})$ while keeping the server computation cost at $O(N)$ homomorphic operations. However, a larger d comes with a higher homomorphic multiplicative depth. As mentioned in Section V-A, it is challenging for SHE schemes to handle high multiplicative depths. More advanced SHE techniques are needed to support a high recursion level d (see the survey below for more details).

D. A Brief Survey of Single-server Computational PIR

Most single-server PIR schemes are based on homomorphic encryption and can be viewed as concrete instantiations of the recursive scheme presented in this section.

Number-theory-based schemes. Kushilevitz and Ostrovsky [41] present the first single-server PIR scheme. Their scheme uses the homomorphic property of the quadratic residue-based public-key encryption of Goldwasser and Micali [42]. Stern [43] gives an oblivious transfer scheme, which is stronger than PIR, based on any *additively* homomorphic encryption (AHE). Chang [44] and Lipmaa [45] present schemes with improved communication costs based on the Damgård-Jurik [46] AHE.

It should be noted that AHE, as the name suggests, only supports the homomorphic ciphertext addition operation CtAdd, and does not support the ciphertext multiplication operation CtMul . But the above AHE-based PIR schemes can still be understood in the SHE-based template outlined in this section. First, note that AHE supports the PtCtMul operation: it can be implemented by invoking CtAdd repeatedly (using the repeated squaring algorithm). The CtMul operation can then be emulated like PtCtMul after interpreting one of the ciphertexts as a plaintext. The server's response of these AHE-based PIR schemes would be a ciphertext with layers of nested encryption.

We are aware of two schemes that do not follow the homomorphic encryption approach [47], [48]. In particular, the scheme of Gentry and Ramzan [48] has the smallest communication cost among all single-server PIR schemes to our knowledge. Its communication cost can approach the insecure baseline (i.e., simply fetching the desired entry) on databases with reasonably large entries.

The above schemes are all based on number-theoretic techniques. The server computation involves a large number of modular exponentiations and is hence very slow. For a more detailed overview of single-server PIR schemes based on number-theoretic techniques, we refer the reader to a survey by Ostrovsky and Skeith [49].

Lattice-based schemes. More recent works [7], [30]–[33], [50]–[53] move away from number-theoretic techniques and instead use lattice-based (in particular, RLWE-based) SHE. Lattice-based SHE at low multiplicative depth offers much cheaper homomorphic operations than number-theoretic AHE, thereby drastically speeding up server computation.

Client

Input: x, the desired index

$x_1 \leftarrow \left\lceil x/\sqrt{N} \right\rceil$ // $\mathsf{DB}[x] \in \mathsf{DB}_{x_1}$

$x_2 \leftarrow x - (x_1 - 1)\sqrt{N}$ // $\mathsf{DB}[x] = \mathsf{DB}_{x_1}[x_2]$

$\mathsf{q}_1, \mathsf{q}_2 \leftarrow$ two vectors of $\sqrt{N}$ ciphertexts

$\mathsf{q}_1[x_1] \leftarrow \mathsf{Enc}(s, 1)$

$\mathsf{q}_1[i] \leftarrow \mathsf{Enc}(s, 0)$, for all $i \neq x_1$

$\mathsf{q}_2[x_2] \leftarrow \mathsf{Enc}(s, 1)$

$\mathsf{q}_2[i] \leftarrow \mathsf{Enc}(s, 0)$, for all $i \neq x_2$

$\mathsf{q}_1, \mathsf{q}_2 \longrightarrow$

Server

Input: DB, the database

$\mathsf{DB}_i \leftarrow \left(\mathsf{DB}[(i-1)\sqrt{N}+1], \ldots, \mathsf{DB}[i\sqrt{N}]\right), \forall i \in [\sqrt{N}]$

$\mathbf{t} \leftarrow \sum_{i=1}^{\sqrt{N}} \mathsf{PtCtMul}(\mathsf{DB}_i, \mathsf{q}_1[i])$

$\mathsf{r} \leftarrow \sum_{j=1}^{\sqrt{N}} \mathsf{CtMul}(\mathbf{t}[j], \mathsf{q}_2[j])$

$\longleftarrow \mathsf{r}$

Return $\mathsf{Dec}(s, \mathsf{r})$

Algorithm 5. Single-Server PIR based on SHE and two levels of recursion. s is the client's secret key. The additions in the summations use CtAdd.

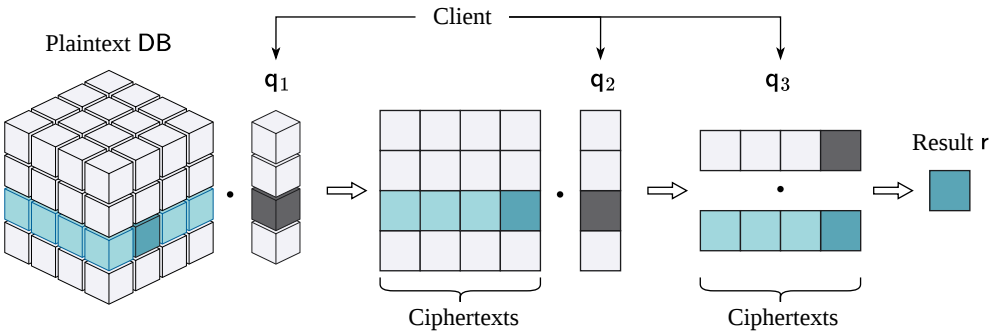


Fig. 4. An illustration for a single-server PIR scheme with three levels of recursion.

The shift to lattice-based SHE also brings new challenges. First, a lattice ciphertext is very large (at least a few thousand bytes) even if the plaintext is small. This is especially wasteful for encrypting the bits in q one by one. Second, homomorphic multiplication introduces large noise, limiting the number of PIR recursion levels. As a result, the initial attempt [30] improves server computation but suffers from high communication cost.

Various techniques have been invented or incorporated to address these two challenges. To address the first challenge, SealPIR [31] introduces query compression: the client can pack many bits in a single lattice ciphertext, and the server can unpack them into individual ciphertexts. This technique significantly reduces the query size. MulPIR [7] incorporates modulus switching [38], [54] to reduce response size. To address the second challenge, several works [32], [50], [51] incorporate a special low-noise homomorphic multiplication [55] to implement the CtMul operation. This allows a higher recursion level d, resulting in smaller query and response sizes. These and additional techniques, along with engineering efforts, help single-server PIR schemes eventually achieve reasonable practical efficiency.

It should be noted that the implementations of most RLWE-based single-server PIR schemes [31]–[33], [52], [56] employ a form of server preprocessing to speed up server computation during the query. In particular, they perform Number-Theoretic Transform (NTT) on each plaintext database entry individually in preparation for the PtCtMul operations. However, the core algorithms of these schemes follow the templates in Algorithms 4 and 5 and do not rely fundamentally on preprocessing. Thus, we treat them as PIR schemes without preprocessing and discuss them in this section. In contrast, the next section

discusses PIR schemes whose design principle critically relies on server preprocessing.

VI. Server-Preprocessing PIR

All PIR schemes (without preprocessing), including those in the previous sections, have server computation costs that are linear in the database size, since the server operates on every single database entry. This is a fundamental barrier because, intuitively, if a PIR scheme does not require the server to operate on a given entry, then the server learns that the client is not retrieving that entry, thereby violating PIR privacy. This intuition was formalized by Beimel, Ishai, and Malkin [57]. They proved the lower bound that $\Omega(N)$ server computation is necessary if the server stores only the unmodified database (and nothing else) and the client has no state.

The above linear server computation lower bound can be circumvented by *preprocessing* the database. Note that any nontrivial PIR scheme (with or without preprocessing) must achieve sublinear communication (possibly in an amortized sense). Naturally, many PIR schemes that use preprocessing aim to make communication and server computation both sublinear in N, so they have also been called *doubly-efficient* PIR. But it is also important to note that not all PIR schemes with preprocessing aim to be doubly efficient. In particular, the scheme we present later in this section still has $O(N)$ server computation, but with an improved constant factor.

If the server carries out the preprocessing without involving any client, then it is called **server-preprocessing PIR**. If preprocessing needs to be carried out separately by each client or for each client, it is called **client-preprocessing PIR**. For both paradigms, *updates* to the database pose new challenges, since they likely require redoing a significant portion, if not the entirety, of the preprocessing step. We discuss server-preprocessing PIR in the rest of this section and client-preprocessing PIR in the next section.

In a server-preprocessing PIR scheme, the server preprocesses the original database in an offline phase and stores the preprocessed database to respond to client queries in an online phase. The stored preprocessed database is inevitably larger than the original database.

While there are a few doubly efficient server-preprocessing PIR schemes in the literature [57], [58], they rely on advanced tools and are beyond the scope of this article. Thus, in this section, we describe a single-server server-preprocessing PIR scheme whose online server computation is still $O(N)$ but a constant factor cheaper than the schemes mentioned in Section V.

A. Background on LWE and RLWE Encryptions

As briefly mentioned in Section V, state-of-the-art single-server PIR schemes without preprocessing use Somewhat Homomorphic Encryption (SHE) based on Ring Learning with Errors (RLWE) [35]–[37]. A more basic and less efficient variant of SHE is based on Learning with Errors (LWE) [34]. Yet, a recent line of work on server-preprocessing PIR [59]–[62] uses LWE-based SHE. Their main observation is that LWE is more amenable to preprocessing. To understand the reasoning here, we need to dive a little deeper into LWE-based and RLWE-based SHE.

In LWE encryption, a plaintext m is a scalar, and a ciphertext has the form $\widetilde{m} = (\mathbf{a}, b)$, where $\mathbf{a}$ is a random vector and b is a scalar. Here, $b = \mathbf{a} \cdot \mathbf{s} + \Delta m + e$, where the vector $\mathbf{s}$ is the secret key, e is a small random error, and Δ is a scaling factor that places m in high-order bits. Decryption works by computing $b - \mathbf{a} \cdot \mathbf{s}$ and rounding to the nearest multiple of Δ to remove the error e.

In RLWE encryption, a plaintext $\mathbf{m}$ is a polynomial, and a ciphertext has the form $\widetilde{\mathbf{m}} = (\mathbf{a}, \mathbf{b})$ where $\mathbf{a}$ and $\mathbf{b}$ are both polynomials. Here, similarly to LWE, $\mathbf{a}$ is a random polynomial and $\mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \Delta \mathbf{m} + \mathbf{e}$, where $\mathbf{s}$ is the secret key polynomial, $\mathbf{e}$ is a small random error polynomial, and Δ is the scaling factor. Decryption works similarly by computing $\mathbf{b} - \mathbf{a} \cdot \mathbf{s}$ and rounding to remove the error $\mathbf{e}$.

The vector length in LWE and the polynomial degree in RLWE are both denoted by n and typically range between 512 and 4096 for PIR. The bit-length of the scalar in LWE and the polynomial coefficients in RLWE must be carefully chosen based on n.

Computation efficiency comparison. Note that an LWE ciphertext is more than n times larger than its plaintext, whereas an RLWE ciphertext is only a constant factor larger than its plaintext. This makes an RLWE-based SHE much more efficient than an LWE-based one in terms of both communication and computation. The communication advantage stems from the constant ratio between the ciphertext and plaintext sizes. To understand the computational advantage of RLWE, let us examine the computation involved in PtCtMul, since that is the server computation bottleneck in Algorithm 5. With LWE,

$$\mathsf{PtCtMul}(m, \widetilde{x}) = m \cdot (\mathbf{a}, b) = (m \cdot \mathbf{a}, m \cdot b)$$

requires $O(n)$ computation where the bottleneck is the scaling of the vector $\mathbf{a}$. With RLWE, the computation involves two polynomial multiplications:

$$\mathsf{PtCtMul}(\mathbf{m}, \widetilde{\mathbf{x}}) = \mathbf{m} \cdot (\mathbf{a}, \mathbf{b}) = (\mathbf{m} \cdot \mathbf{a}, \mathbf{m} \cdot \mathbf{b}).$$

If the RLWE ciphertext is kept in the Number-Theoretic Transform (NTT) form, the computation is also $O(n)$. But notice that the underlying plaintext of an RLWE ciphertext is thousands of times bigger than that of an LWE ciphertext. In other words, while the computation cost of PtCtMul is similar with LWE and RLWE, the RLWE PtCtMul processes thousands of times more plaintext data in each operation, and hence boasts much superior computational efficiency.

Computation efficiency comparison with preprocessing. Preprocessing changes the above comparison. Note that in both LWE and RLWE ciphertexts, only the second component depends on the plaintext. The first component $\mathbf{a}$ is purely random and independent of the plaintext. In the context of PIR, the plaintext operand is the database, which is known in advance. Then, the computation involving the $\mathbf{a}$ component can be precomputed in an offline preprocessing phase.

For RLWE-based SHE, this observation is immaterial because the computation involving $\mathbf{a}$ is only half of the total

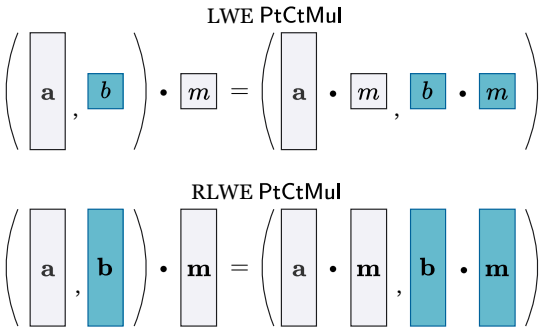


Fig. 5. An illustration of LWE-based and RLWE-based PtCtMul. The light-colored terms are fixed in advance and can be precomputed in the offline phase. The dark-colored terms require online computations.

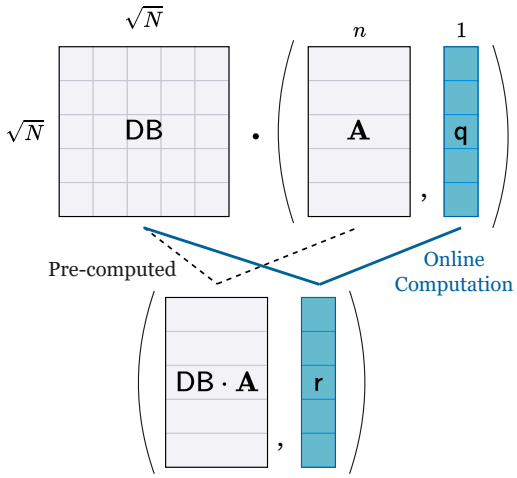


Fig. 6. Separating the first dimension inner product into online and offline phases. The light-colored terms are fixed in advance and are multiplied in the offline phase. The dark-colored terms require online computations.

cost. For LWE-based SHE, however, this observation makes a huge difference. The dominant part of the computation for LWE-based $\mathsf{PtCtMul}(m, \mathbf{a})$ can now be precomputed. The remaining online computation ($m \cdot b$) is a single scalar multiplication. This helps LWE-based SHE match RLWE-based SHE asymptotically in terms of *online* computation, when normalized by the amount of plaintext data. In fact, the online computation efficiency of LWE-based SHE is now a constant factor (about $4\times$) better than that of RLWE-based SHE, because it does not need NTT.

This observation gives rise to several single-server server-preprocessing PIR schemes [59]–[62] with improved (though still linear in N) online server computation. We next explain their high-level ideas.

B. Single-Server Server-Preprocessing PIR based on LWE

The overall structure of these schemes is similar to that of Algorithm 5. In Algorithm 5, the inner product between DB' and q_1 using PtCtMul dominates the online server computation. The main idea is to use an LWE-based SHE for the first dimension to improve the online server computation.

The query vector q_1 consists of $\sqrt{N}$ ciphertexts. With LWE-based SHE,

$$\mathsf{q}_1 = \begin{bmatrix} \mathbf{a}_1 & b_1 \\ \mathbf{a}_2 & b_2 \\ \vdots & \vdots \\ \mathbf{a}_{\sqrt{N}} & b_{\sqrt{N}} \end{bmatrix}$$

The above can be written as $\mathsf{q}_1 = (\mathbf{A}, \mathbf{b})$, where $\mathbf{A}$ is a $\sqrt{N}$-by-n matrix and $\mathbf{b}$ is a column vector of length $\sqrt{N}$. The inner products can now be expressed as matrix multiplication:

$$\mathsf{DB} \cdot \mathsf{q}_1 = \mathsf{DB} \cdot (\mathbf{A}, \mathbf{b}) = (\mathsf{DB} \cdot \mathbf{A}, \mathsf{DB} \cdot \mathbf{b}).$$

As illustrated in Figure 6, *if* both DB and $\mathbf{A}$ are fixed in advance, then the server can precompute $\mathsf{DB} \cdot \mathbf{A}$ and reuse the result in all online PIR queries from all clients. Here, it is assumed that the database DB rarely changes (a common assumption in server-preprocessing PIR). The matrix $\mathbf{A}$, which consists of the first components of LWE ciphertexts, is typically generated randomly on-the-fly by each client. In order to fix $\mathbf{A}$, the server can generate it pseudorandomly from a short random seed. The seed is considered part of the database metadata that each client obtains before making PIR queries. This way, every client can reproduce and use the same matrix $\mathbf{A}$ in all their PIR queries.

Algorithm 6 describes a single-server server-preprocessing PIR scheme following this paradigm. The database is structured as a square matrix in column-major order. The client generates two encrypted one-hot vectors, corresponding to the column and row of the desired index in the square matrix.

Crucially, the first one-hot vector is encrypted using LWE. As discussed, the same pseudorandom matrix $\mathbf{A}$ is generated from the same public seed by the server and all clients on all queries. We note that the client need not include $\mathbf{A}$ in the PIR query since the server already has it; only the plaintext-dependent term $\mathbf{b}$ needs to be sent.

Reusing $\mathbf{A}$ across queries warrants some discussion. In LWE encryption, security critically depends on either the first component $\mathbf{a}$ or the secret key $\mathbf{s}$ being random. Now that $\mathbf{A}$ is fixed, the client in Algorithm 6 must use a fresh secret key $\mathbf{s}$ for each PIR query. (In contrast, the client in Algorithm 5 can use either a fresh or a long-term secret key.)

The server evaluates the first inner product, which is expressed as a matrix multiplication. As discussed, the bulk of the computation arises from $\mathsf{DB} \cdot \mathbf{A}$ and takes place in the offline preprocessing phase. After this step, the server holds a column of the database encrypted with LWE.

From here, there are several design choices. First, one could skip the second dimension altogether and send back an entire encrypted column (after some compression) [59], [60]. This design incurs a large response size. Algorithm 6 leaves out this option and instead assumes a second inner product will be performed. There are at least two approaches to this second inner product. One option is to reinterpret the LWE ciphertexts as plaintexts and repeat the LWE-based matrix multiplication [61]. Another option is to switch to RLWE for the second dimension, which requires converting LWE ciphertexts into RLWE ciphertexts [62]. The rest of Algorithm 6 is similar to Algorithm 5. The server sends back

Client		**Server**
Input: x, the desired index		**Input:** DB, the database
		View DB as a $\sqrt{N}$-by-$\sqrt{N}$ matrix in column-major order
		$\mathbf{A} \leftarrow \sqrt{N}$-by-$n$ pseudorandom matrix generated from a public seed
		$\mathbf{A}' \leftarrow \mathsf{DB} \cdot \mathbf{A}$
..... Start online phase for one query		
$x_1 \leftarrow \left\lceil x/\sqrt{N} \right\rceil$ // column of x		
$x_2 \leftarrow x - (x_1 - 1)\sqrt{N}$ // row of x		
$\mathbf{m}_1, \mathbf{m}_2 \leftarrow$ one-hot encodings of x_1, x_2		
Regenerate $\mathbf{A}$ from the public seed		
Sample a fresh secret key $\mathbf{s}$		
$\mathbf{b} \leftarrow \mathbf{A} \cdot \mathbf{s} + \Delta\mathbf{m}_1 + \mathbf{e}$ // LWE-based SHE		
$\mathsf{q}_2 \leftarrow \mathsf{Enc}(\mathbf{s}, \mathbf{m}_2)$ // either LWE-based or RLWE-based SHE		
	$\xrightarrow{\mathbf{b}, \mathsf{q}_2}$	
		$\mathbf{b}' \leftarrow \mathsf{DB} \cdot \mathbf{b}$ // First dimension inner product
		$\mathsf{r} \leftarrow$ homomorphic inner product between $(\mathbf{A}', \mathbf{b}')$ and q_2
	$\xleftarrow{\mathsf{r}}$	
Return $\mathsf{Dec}(\mathbf{s}, \mathsf{r})$		

Algorithm 6. Server-preprocessing single-server PIR with LWE-based SHE and two levels of recursion.

the encrypted result after the second-dimension homomorphic inner product. The client decrypts and reconstructs the desired entry. (If RLWE is used for the second dimension, the client's LWE secret key is reinterpreted as an RLWE secret key [63].)

Efficiency. In the offline phase, the server precomputes the matrix multiplication $\mathsf{DB} \cdot \mathbf{A}$. A straightforward implementation requires $O(Nn)$ computation. A more sophisticated approach uses a structured matrix $\mathbf{A}$ along with RLWE techniques to reduce the offline computation to $O(N \log n)$ [61]. If the second dimension uses RLWE, additional preprocessing is required to prepare for the LWE-to-RLWE conversions.

It is worth noting that updating a single database entry requires redoing part of the preprocessing (or even the entire preprocessing in some advanced schemes [61], [62]). In general, PIR schemes based on preprocessing are not well-suited for databases with frequent updates.

The online communication cost is $O(\sqrt{N})$. The online server computation is $O(N)$ for sufficiently large databases, since the first-dimension matrix multiplication touches every database entry. The second dimension may also contribute substantially to the online server computation cost. For example, the LWE-to-RLWE conversions in [62] require $O(\sqrt{N}n)$ online server computation.

Finally, the scheme can require nontrivial client computation. The client needs to compute $\mathbf{A} \cdot \mathbf{s}$ on every query. These steps lead to $O(\sqrt{N}n)$ online client computation in a basic implementation. Using a structured matrix $\mathbf{A}$ and RLWE techniques can again reduce this cost to $O(\sqrt{N} \log n)$ [61].

C. A Brief Survey of Server-Preprocessing PIR

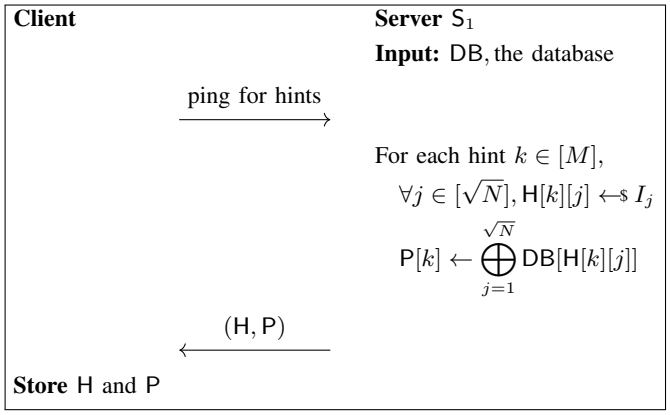
PIR with preprocessing was first studied in the aforementioned work of Beimel, Ishai, and Malkin [57]. They give a family of multi-server server-preprocessing PIR schemes that, with $k \geq 2$ servers, achieve $O(N^{\frac{1}{2k-1}})$ communication and $O(N/(\epsilon \log N)^{2k-2})$ server computation, while requiring $O(N^{1+\epsilon})$ server storage.

Lin, Mook, and Wichs [58] construct the first doubly efficient single-server server-preprocessing scheme. For any constant $\epsilon > 0$, their scheme requires $O(N^{1+\epsilon})$ preprocessing time and server storage, and achieves $\mathsf{poly}(\lambda, \log^{1+2/\epsilon} N)$ online communication and computation per query. Although the asymptotics look appealing, the scheme is still far from practical [64].

Inspired by the observation that LWE is preprocessing-friendly, a line of work [59]–[62] proposes single-server server-preprocessing schemes whose online server computation is still $O(N)$ but cheaper than RLWE-based schemes. The scheme we presented in this section is a simplified version of this line of work.

We next mention a few works that do not fit cleanly into our classifications.

Boyle et al. [65] and Canetti et al. [66] first explored doubly efficient single-server PIR. They first consider a client-preprocessing setting in which each client downloads and preprocesses the database, but the preprocessed database is stored back on the server. Note that in this model, the server needs to store a separate copy of the database for each client, and each copy is asymptotically larger than the original database. This is at odds with the *public database* nature of PIR (see the discussion in Section II-D on ORAM). Boyle



Algorithm 7. The offline preprocessing phase of the two-server stateful PIR scheme. $M = \lambda\sqrt{N}$ hints are sampled where λ is a security parameter.

et al. [65] also proposed a heuristic transformation of their scheme so that a trusted third party performs preprocessing and the server stores a single (preprocessed) copy of the database. In addition to the limitations in the models, these two works rely on less-studied cryptographic assumptions and heavyweight theoretical tools.

SimplePIR [67] and FrodoPIR [68] are the first to observe that the bulk of the computation in LWE-based SHE can be offloaded to a preprocessing phase, so that the online computation is $O(N)$ but cheaper than RLWE. In these two schemes, the preprocessing computation is performed entirely by the server, but each client needs to download and store the preprocessing result. The client storage required is $O(\sqrt{N}n)$, where n is the LWE vector length, typically 1024. Thus, these two schemes are more suitable for databases with large N such that $\sqrt{N}n \ll N$.

VII. Stateful Client-preprocessing PIR

In many client-preprocessing PIR schemes, the client stores some preprocessed information locally, so these schemes are also called *stateful* PIR schemes. Most stateful PIR schemes are doubly efficient, i.e., their amortized communication and computation costs are both sublinear in N. We describe a simple two-server stateful PIR scheme in this section.

A. Problem Definition

A stateful client-preprocessing PIR scheme consists of two phases: an *offline preprocessing* phase and an *online query* phase.

1) In the offline preprocessing phase, the client interacts with the server(s) to obtain some *hints* about the database. The hints are stored locally at the client.
2) In the online query phase, the client utilizes the stored hints to perform queries. The interaction in this phase is similar to PIR without preprocessing in Figure 1: the client sends a query, receives a response, and computes the answer.

The hints are specific to one client. In other words, every client must perform the offline preprocessing phase separately with the server(s). Needless to say, the hints stored by a client must be smaller than the database for the scheme to be nontrivial.

The offline preprocessing phase inevitably involves computation linear (and often superlinear) in the database size. Thus, a stateful PIR scheme shows its benefits only if the same client makes many queries to the database during the online query phase. (In comparison, a server-preprocessing PIR scheme can show benefits after many clients each make one query.)

Correctness and privacy. Note that correctness and privacy are defined for a single query in Section II. That is appropriate for standard PIR (without preprocessing) because the client there is stateless, and the server's state does not change. Thus, even if the client makes multiple queries, the correctness and privacy of a query are clearly independent of other queries.

For a stateful PIR, however, that is often not the case, as the client state may create dependency between queries. We extend the correctness and privacy definitions in Section II from a single query to a sequence of queries. The client wishes to retrieve a sequence of desired entries whose indices are $[x_1, x_2, \ldots, x_m]$. The client does not have to come up with the sequence upfront. An index in the sequence can depend on the *answers* to the previous queries.

- **Correctness:** For a sequence of indices $[x_1, x_2, \ldots, x_m]$, the i-th answer computed by the client must match the x_i-th entry in the database.
- **Privacy:** The server learns no information about the client's desired indices from the sequence of the client's queries. The indistinguishability game for privacy from Section II extends naturally to two sequences of indices.

B. A Two-server Stateful PIR Scheme

We next describe a simple two-server stateful PIR scheme. The two servers S_1 and S_2 each hold a copy of the database DB. Looking ahead, the client will retrieve hints from S_1 and use those hints to perform queries with S_2.

Let the database DB consist of N entries indexed by integers in $[N]$. We partition the database DB into $\sqrt{N}$ partitions, each containing $\sqrt{N}$ consecutive entries. Concretely, the j-th partition for each $j \in [\sqrt{N}]$ contains the entries at indices $I_j = \left\{(j-1)\sqrt{N}+1, (j-1)\sqrt{N}+2, \ldots, j\sqrt{N}\right\}$.

Looking ahead, the scheme often samples a uniformly random index from a partition. We write $r \leftarrow\!\$\ I_j$ to mean that r is an index sampled uniformly randomly from the j-th partition.

The offline preprocessing phase. During the offline preprocessing phase, S_1 samples M hints and sends them to the client. The value of M will be specified later. Each hint consists of $\sqrt{N}$ randomly sampled indices, one from each partition, and the parity (i.e., bitwise XOR) of the entries at those indices. Precisely, the k-th hint is $(\mathsf{H}[k], \mathsf{P}[k])$ where:

- $\mathsf{H}[k]$ is a list of $\sqrt{N}$ indices where $\mathsf{H}[k][j] \leftarrow\!\$\ I_j$ is a random index from the j-th partition, and
- $\mathsf{P}[k] = \bigoplus_{h \in \mathsf{H}[k]} \mathsf{DB}[h]$ is a parity of the entries at the indices in $\mathsf{H}[k]$.

Server S_1 then sends H and P to the client. The client stores H and P locally. The pseudocode of the offline preprocessing phase is given in Algorithm 7.

Observe that an entry can be recovered from the parity and *all the other* entries contributing to the parity, i.e., if $x \in \mathsf{H}[k]$,

$$\mathsf{DB}[x] = \mathsf{P}[k] \oplus \left(\bigoplus_{h \in \mathsf{H}[k] \setminus \{x\}} \mathsf{DB}[h] \right).$$

This observation is the central idea of most stateful PIR schemes in the literature. We will rely on it to retrieve entries during the online query phase.

The online query phase. Given a desired index x, the client computes the partition ℓ containing the desired index as:

$$\ell = \left\lceil \frac{x}{\sqrt{N}} \right\rceil.$$

Then, the client finds a hint whose set of indices contains x, i.e., the client scans the set of hints to find a hint k such that

$$\mathsf{H}[k][\ell] = x.$$

The parameter M mentioned in the offline phase will be set to ensure that such a hint exists with overwhelming probability. We call hint k the *selected hint*.

The client then constructs a query q consisting of $\sqrt{N}$ indices such that for every $j \in [\sqrt{N}]$,

$$\mathsf{q}[j] = \begin{cases} \mathsf{H}[k][j], & j \neq \ell, \\ \alpha_\ell \leftarrow\!\$\, I_\ell, & j = \ell. \end{cases}$$

In other words, the query q consists of the indices in the selected hint $\mathsf{H}[k]$, except that $\mathsf{H}[k][\ell]$ (which is the desired index x) is replaced with a random index from partition ℓ.

Note that it is critical to replace $\mathsf{H}[k][\ell]$. If the client simply sends all the indices in $\mathsf{H}[k]$, it is clearly insecure, since the desired index x always appears in the query q, allowing the server to narrow it down to the set of indices in q. With the replacement, all the indices in q are random.

The client sends q to server S_2. S_2 returns the database entries at the indices in q. Specifically, the response r consists of $\sqrt{N}$ database entries such that for all $j \in [\sqrt{N}]$,

$$\mathsf{r}[j] = \mathsf{DB}[\mathsf{q}[j]].$$

Finally, the client computes

$$\left(\bigoplus_{j \in [\sqrt{N}] \setminus \{\ell\}} \mathsf{r}[j] \right) \oplus \mathsf{P}[k]$$

as the answer. This uses the previous observation that the desired entry can be recovered from the parity and the other entries involved in the parity.

The pseudocode and a pictorial illustration for constructing the query can be found in the top part of Algorithm 8 and Figure 7.

Hint replenishment. The selected hint k cannot be reused for subsequent queries because the server has already seen (most of) the indices in that hint. As it turns out, it would also be incorrect and insecure if the client simply discarded the used hint k and continued making subsequent queries. Consider a situation in which the client repeatedly queries the same desired index x. The selected hints for all these queries must contain the desired index x. If the used hints are simply discarded, all the hints containing x will soon be depleted, and subsequent queries for x can no longer be fulfilled, violating correctness. The reason why privacy would also be affected is more subtle. Once many hints containing x are consumed (even before they are depleted), the server can observe that the index x appears in subsequent client queries with a lower frequency than other indices, thereby revealing to the server that the client likely queried x previously.

Therefore, after the client consumes a selected hint to complete a query, the client needs to carry out a procedure to replace the consumed hint with a new hint. The new hint must follow the same distribution as the consumed hint, i.e., containing the desired index x from partition ℓ and random indices from all the other partitions. We call this procedure *hint replenishment*. It can be done as follows.

The client pings server S_1 to request a new hint. S_1 samples a new hint for the client, like what it did during the offline preprocessing phase: it samples one random index from each partition, i.e., a vector h where $\forall j \in [\sqrt{N}]$, $\mathsf{h}[j] \leftarrow\!\$\ \ I_j$. Furthermore, S_1 sends the client the database entries at those indices, i.e., $\mathsf{DB}[\mathsf{h}[j]]$ for each $j \in [\sqrt{N}]$.

The client updates the consumed hint k as follows:

$$\mathsf{H}[k][j] = \begin{cases} \mathsf{h}[j], & j \neq \ell \\ x, & j = \ell \end{cases}$$

In other words, the client replaces the ℓ-th index in the hint with the desired index x that it just queried. This modifies a regular hint h into one that follows the same distribution as the consumed hint, i.e., containing the desired index x from partition ℓ and random indices from all the other partitions.

The client now needs to compute the parity for this modified hint, which should be

$$\mathsf{P}[k] = \left(\bigoplus_{j=1}^{\ell-1} \mathsf{DB}[\mathsf{h}[j]] \right) \oplus \mathsf{DB}[x] \oplus \left(\bigoplus_{j=\ell+1}^{\sqrt{N}} \mathsf{DB}[\mathsf{h}[j]] \right).$$

$\mathsf{DB}[x]$ has just been retrieved by the client. Each $\mathsf{DB}[\mathsf{h}[j]]$ is sent to the client by S_1. The client can thus easily compute the above parity.

The pseudocode and a pictorial illustration of the hint replenishment procedure can be found in the bottom part of Algorithm 8 and Figure 7.

C. Correctness, Privacy, and Efficiency

We provide a proof sketch for the correctness and privacy of the scheme.

Correctness. Let us first consider the very first query. The client correctly retrieves the desired entry if it finds a hint that contains the desired index x. Since each hint samples a random index per partition, the probability that a hint contains the desired index is $\frac{1}{\sqrt{N}}$. Since hints are sampled independently of each other, if the client samples $M = \lambda\sqrt{N}$ hints during the offline preprocessing phase, then the probability that none of the M hints contains x is:

$$\left(1 - \frac{1}{\sqrt{N}}\right)^{\lambda\sqrt{N}} < e^{-\lambda}.$$

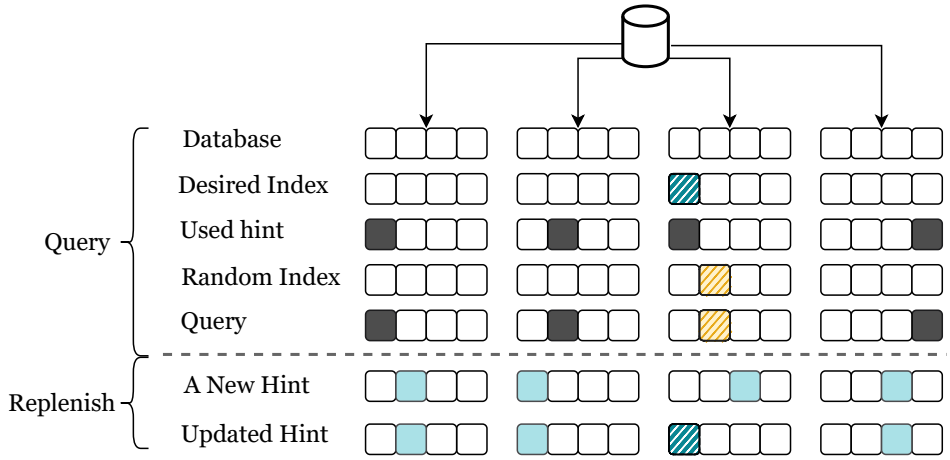


Fig. 7. An illustration for the query construction and hint replenishment in the two-server stateful PIR scheme.

Client

Input: x, the desired index

$\ell \leftarrow \left\lceil x/\sqrt{N} \right\rceil$

Find k such that $\mathsf{H}[k][\ell] = x$

$\mathsf{q} \leftarrow \mathsf{H}[k]$

$\mathsf{q}[\ell] \leftarrow\!\!\$\ I_\ell$

$\xrightarrow{\mathsf{q}}$

Server S_2

Input: DB, the database

$\forall j \in [\sqrt{N}], \mathsf{r}[j] \leftarrow \mathsf{DB}[\mathsf{q}[j]]$

$\xleftarrow{\mathsf{r}}$

$$\mathsf{a} \leftarrow \mathsf{P}[k] \oplus \left(\bigoplus_{j=1}^{\ell-1} \mathsf{r}[j]\right) \oplus \left(\bigoplus_{j=\ell+1}^{\sqrt{N}} \mathsf{r}[j]\right)$$

Server S_1

Input: DB, the database

$\xrightarrow{\text{ping to replenish a hint}}$

$\forall j \in [\sqrt{N}], \mathsf{h}[j] \leftarrow\!\!\$\ I_j$

$\forall j \in [\sqrt{N}], \mathsf{rh}[j] \leftarrow \mathsf{DB}[\mathsf{h}[j]]$

$\xleftarrow{(\mathsf{h}, \mathsf{rh})}$

$\mathsf{H}[k] \leftarrow \mathsf{h}$

$\mathsf{H}[k][\ell] \leftarrow x$

$$\mathsf{P}[k] = \left(\bigoplus_{j=1}^{\ell-1} \mathsf{rh}[j]\right) \oplus \mathsf{DB}[x] \oplus \left(\bigoplus_{j=\ell+1}^{\sqrt{N}} \mathsf{rh}[j]\right)$$

Return a

Algorithm 8. The online query phase with hint replenishment of the two-server stateful PIR scheme. The hint indices H and hint parities P are initialized by Algorithm 7 and stored by the client across queries.

Setting M to $O(\lambda\sqrt{N})$ makes the probability of correctness failure exponentially small in the security parameter λ.

To extend this analysis to subsequent queries, we observe that the hint distribution remains the same before and after each query because the hint replenishment procedure replenishes a hint that follows the same distribution as the consumed hint. Then, following the same argument made for the first query, the client will be able to find suitable hints to fulfill all subsequent queries except for an exponentially small probability.

Privacy. Privacy against server S_1 is straightforward since the client only receives hints from S_1 and never sends any information to S_1. We focus on privacy against server S_2. Let us again look at the very first query. The selected hint contains the desired index and random indices from other partitions. The query q then replaces the desired index in the selected hint with a random index from the same partition. Hence, the query q sent to S_2 consists of one random index from each partition, independent of the desired index. Therefore, S_2 learns no information from q about the desired index.

Similarly to the correctness proof, the privacy proof can be extended to subsequent queries by showing that the hint distribution, even conditioned on the view of S_2 during a query, remains unchanged before and after each query. More precisely, the set of hints remains identically distributed, since each consumed hint is replaced by a newly sampled hint that follows the same distribution.

Client storage. A keen reader may have noticed that the client storage of the presented scheme is superlinear in N. In particular, the client storage is $O(\lambda N \log N + \lambda\sqrt{N}w)$ where w is the size of each database entry in bits. This storage comes from the $M = O(\lambda\sqrt{N})$ hints, where each hint comprises $\sqrt{N}$ random indices of $O(\log N)$ bits each and a parity of w bits. When the entry size w is smaller than $O(\lambda \log N)$, the client storage in the presented scheme can be even larger than the entire database, which is clearly unreasonable.

Fortunately, the superlinear client storage is from the randomly sampled indices. We can use a *Pseudorandom Function* (PRF) to generate all the indices in all the hints pseudorandomly from a single random seed of $O(\lambda)$ bits. Some care is needed when dealing with replenished hints. Recall that the client receives a (now PRF-generated) hint from the server S_1 but then overwrites one of its indices with the just queried desired index to obtain the replenished hint. The client needs to keep track of this change for each replenished hint and act accordingly during hint selection and query construction.

After applying PRF and accompanying modifications, the scheme uses $O(\lambda\sqrt{N}\log N + \lambda\sqrt{N}w)$ bits of client storage, which is sublinear in N and smaller than the database for most realistic values of N. A trade-off is that the scheme's privacy guarantee is downgraded from information-theoretic to computational.

Communication and Computation. The offline preprocessing phase is run only once, after which the client can make unlimited queries in the online phase. Thus, after sufficiently many queries, the offline preprocessing cost becomes insignificant in the *amortized* sense. The amortized cost of the stateful PIR scheme is thus just the cost of the online query phase.

During each query, the client sends $\sqrt{N}$ indices to S_2 and receives $\sqrt{N}$ entries. Furthermore, the client receives $\sqrt{N}$ indices and entries from S_1 for a new hint. Hence, the communication cost is $O(\sqrt{N}\log N + \sqrt{N}w)$.

The servers do not perform much additional computation beyond reading the $\sqrt{N}$ entries from the database and sending them to the client. Since the server's computation cost is small, the client's computation should now be analyzed in the stateful PIR paradigm. To select a hint, the client needs to examine $\sqrt{N}$ hints in expectation. If the hint's indices are generated by PRF, this step requires $O(\sqrt{N})$ PRF evaluations in expectation. After receiving the server response, the client computes a parity of $\sqrt{N}$ entries. Depending on the entry size, the client computation bottleneck is either the $O(\sqrt{N})$ PRF evaluation or the $O(\sqrt{N})$ w-bit XORs.

D. Adapting to the Single-server Setting

Many two-server stateful schemes, including the one we just presented, can be adapted to the single-server setting.

With only a single server, the client has to fetch hints from the same server it sends queries to. Thus, the hints need to be fetched from the server in a private manner, without the server learning any information about them. This can be done by streaming the entire database to the client during the offline preprocessing phase. While streaming the database is costly and inelegant, attempts to fetch hints privately without it [32], [69], [70] have not yielded practical solutions.

In addition to the main hints, the client also needs to fetch some *backup hints* in the preprocessing phase [69]. The backup hints are used to replenish consumed hints after each query. Eventually, the client will run out of backup hints. Thus, the preprocessing phase needs to be executed periodically in the single-server setting (rather than only once at the beginning with two non-colluding servers).

It is worth noting that many existing single-server stateful PIR schemes can achieve sublinear (in N) amortized communication from only one-way functions. (A recent work even achieves this without any cryptographic assumptions [71].) In contrast, single-server PIR with sublinear communication without preprocessing is believed to require cryptographic primitives stronger than one-way functions (Section II-D).

E. A Brief Survey of Sublinear Stateful PIR Schemes

Patel, Persiano, and Yeo [72] introduced the paradigm of stateful client-preprocessing PIR and presented a single-server stateful scheme. Their goal in that work was not to achieve sublinear in N server computation. Instead, it was to replace the $O(N)$ expensive RLWE homomorphic operations with $O(N)$ PRF evaluations for server computation. The client in their scheme has to store $O(\sqrt{N})$ hints and perform $O(N)$ PRF evaluations.

Corrigan-Gibbs and Kogan [73] constructed the first stateful PIR scheme with sublinear server computation. In their scheme, the client fetches *hints* from the first server during the offline preprocessing phase, uses the hints to make queries to

the second server, and replenishes a hint from the first server after each query. This blueprint is followed by most subsequent two-server stateful schemes, including the one we presented in this section. Corrigan-Gibbs, Henzinger, and Kogan [69] later adapted the above blueprint to a single server using backup hints.

Yeo proves a lower bound on the space-time trade-off of stateful PIR schemes: $ST = \Omega(N)$ where S is the client storage and T the online server computation [74]. Early schemes [69], [73] have large gaps of $\mathsf{poly}(\lambda, \log N)$ factors from this lower bound. Subsequent works have gradually removed these extra factors. Notably, $ST = O(\lambda N)$ has been achieved with reasonable practical efficiency both with two servers [75] and with a single server [76], [77]. The scheme we presented in this section can be viewed as a simplified version of TreePIR [75], which introduces the partition-based hints and relies on two servers. The remaining λ factor comes from the $M = O(\lambda\sqrt{N})$ hints needed to drive down the correctness failure probability. Building on a new permutation-based hint structure of [78], Wang and Ren [79] construct a single-server scheme that achieves the asymptotically optimal $ST = O(N)$ trade-off.

Two-server stateful schemes have also achieved $\mathsf{poly}\log N$ communication with reasonable practical efficiency [75], [80]. Single-server schemes with $\mathsf{poly}\log N$ communication [81], [82] have thus far relied on theoretical cryptographic tools with no known practical instantiations.

VIII. Keyword PIR

So far in this article, we have assumed that the database entries are identified by integer indices $\{1, 2, \ldots, N\}$. We refer to this standard variant as *index PIR*. In many practical applications, however, the database entries are identified by arbitrary strings or keywords rather than integer indices. For example, in the news-reading example, a news article is often identified by its title or URL. This calls for a variant of PIR called *keyword PIR*.

A. Problem Definition

In keyword PIR, the database DB consists of N key-value pairs, $\mathsf{DB} = \{(k_1, v_1), (k_2, v_2), \ldots, (k_N, v_N)\}$, and the client wants to retrieve the database entry corresponding to a *desired key* x. The interaction between the client and the server stays the same as in Figure 1: the client sends a query, receives a response, and computes the answer. The privacy definitions in Sections II-B and VII-B also extend naturally to keyword PIR. The correctness definition, however, comes with a small modification. If the client's desired key is in the database, the client must retrieve the corresponding value as usual. But if the client's desired key is not present in the database, then the client must learn this fact. More formally,

- **Correctness:** If $(x, v) \in \mathsf{DB}$, then the answer computed by the client equals v; if $(x, v) \notin \mathsf{DB}$, then the answer computed by the client is a special symbol $\perp$.

One can also think of the keyword PIR problem as an index PIR on a *sparse* database. More concretely, suppose that each keyword is n-bit long. One can, of course, treat each keyword as an n-bit integer and solve keyword PIR trivially by running an index PIR on a database of 2^n entries, where the N entries reside at the positions corresponding to their keywords and the other entries (corresponding to the non-existent keywords) are $\perp$. But it is easy to see that this trivial scheme can be extremely costly because its efficiency depends on the keyword space 2^n, which can be orders of magnitude larger than N.

Keyword PIR schemes in the literature have their efficiency depend on N, the number of entries in the database. We will present one such scheme based on a *cuckoo hashing table*.

B. Cuckoo Hash Table

A cuckoo hash table uses several hash functions to insert a set of N key-value pairs into an array CH whose size is slightly larger than N. Typical parameter choices are two hash functions with $|\mathsf{CH}| \approx 2N$ or three hash functions with $|\mathsf{CH}| \approx 1.5N$. The keyword PIR scheme in this section uses the former. Let h_1 and h_2 denote two hash functions that map keys to random positions in CH.

The main invariant of the cuckoo hash table is that a key-value pair (k, v) can only reside in one of the two positions in CH specified by the two hash functions. More concretely, if (k, v) has been inserted, then one of $\mathsf{CH}[\mathsf{h}_1(k)]$ or $\mathsf{CH}[\mathsf{h}_2(k)]$ equals $k\|v$.

Note that CH stores a value with its key prepended. (We assume all the keys are padded to a fixed length.) This ensures there will be no false positives: if no key-value pair with key k has ever been inserted into the cuckoo hash table, then no element in CH starts with k.

With the above invariants, a cuckoo hash table lookup is straightforward: given a key k, simply read the two entries of CH at positions $\mathsf{h}_1(k)$ and $\mathsf{h}_2(k)$. If one of the entries is $k\|v$, then return v as the value; If neither entry starts with the key k, then return $\perp$.

There is a chance that the N key-value pairs cannot all be inserted while respecting the invariant. But the probability of that happening is small. If it happens, one may also retry with a different set of hash functions.

C. Keyword PIR using a Cuckoo Hash Table

Algorithm 9 gives a keyword PIR scheme that uses a cuckoo hash table and invokes an index PIR scheme twice. The same transformation also yields preprocessing/stateful keyword PIR from preprocessing/stateful index PIR.

The server picks two hash functions $\mathsf{h}_1(\cdot)$ and $\mathsf{h}_2(\cdot)$, and inserts every key-value pair in the database DB into the cuckoo hash table to obtain the array CH. We assume the client learns the two hash functions used as part of the database metadata before the PIR protocol starts. Then, given a desired key x, the client essentially performs the cuckoo hash table lookup using index PIR. Concretely, the client makes two index PIR queries in parallel with the desired indices being $\mathsf{h}_1(x)$ and $\mathsf{h}_2(x)$. If one of the two index PIR answers is $x||v$, the keyword PIR returns v. If neither of the two index PIR answers starts with the keyword x, the keyword PIR returns $\perp$.

Privacy follows from the index PIR privacy. Correctness follows from the cuckoo hash table property. It is worth noting

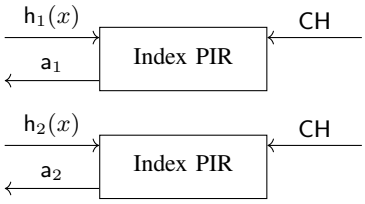
Client

Input: x, desired keyword

Server

Input: DB, the database as key-value pairs

Insert every $(k, v) \in \mathsf{DB}$ into cuckoo hash table CH



If any $\mathsf{a}_j = x||v$, **return** v

Else, **return** $\perp$

Algorithm 9. Keyword PIR from index PIR. The client and the server agree on the two hash functions.

Client

Input: $\{x_1, x_2, \ldots, x_b\}$, desired keywords

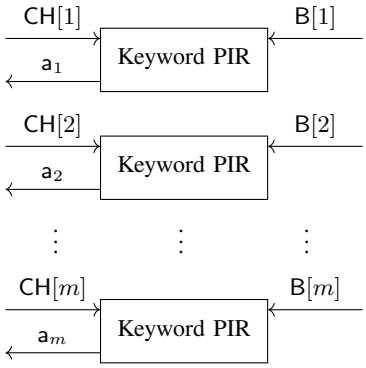
Insert every x_i into cuckoo hash table CH

Server

Input: DB, the database

Initialize m buckets $\mathsf{B}[1], \mathsf{B}[2], \ldots, \mathsf{B}[m]$

Add every $(k, v) \in \mathsf{DB}$ to $\mathsf{B}[\mathsf{h}_1(k)], \mathsf{B}[\mathsf{h}_2(k)], \mathsf{B}[\mathsf{h}_3(k)]$



$\mathsf{a} \leftarrow \{\mathsf{a}_i \mid \mathsf{CH}[i] \text{ is not empty}\}$

Return a

Algorithm 10. Batch PIR using any keyword PIR scheme (e.g., Algorithm 9). The client and the server agree on the three hash functions. The number of buckets is $m = |\mathsf{CH}| \approx 1.5b$.

that the failure probability of the cuckoo hash table does not affect the correctness or privacy of the keyword PIR scheme. If the server fails to insert all entries into the cuckoo hash table, it can simply choose a different set of underlying hash functions and try again. The server can repeat this process until all the entries in the database are successfully inserted. This is why a cuckoo hash table with only two hash functions, despite its higher failure probability, is not only sufficient here but also preferred due to its better efficiency.

The keyword PIR scheme invokes an index PIR scheme twice on a database of size $\approx 2N$. Hence, the communication cost is at most $2\times$ of the underlying index PIR, and the server computation is at most $4\times$ of the underlying index PIR. With additional tricks [83], the server computation can approach $1\times$ the underlying index PIR.

D. A Brief Survey of Keyword PIR

Keyword PIR was first studied in Chor et al. [84]. They gave a black-box transformation from index PIR to keyword PIR, using $\log N$ sequential invocations of an index PIR scheme. The cuckoo-hashing-based keyword PIR presented in this section is known in the folklore and is articulated by Ali et al. [7]. There are also keyword PIR schemes that are not black-box transformations from index PIR [83], [85], [86]. In particular, Patel et al. [83] further reduce the response size to less than $2\times$, at the expense of heavy offline preprocessing and slightly higher online server computation than the cuckoo hash table approach.

IX. Batch PIR

In the PIR variants discussed so far, we have assumed that the client has a *single* entry to retrieve from the server. In some applications, the client may want to retrieve a batch of entries from the server. We call this variant *batch PIR*.

A. Problem Definition

In batch PIR, the client has a set of b desired indices or keywords $\{x_1, x_2, x_3, \ldots, x_b\}$ and wants to retrieve all b entries from DB corresponding to these indices or keywords. The interface and correctness/privacy definitions from Section II extend naturally to batch PIR: the client obtains all b entries

and no information is revealed about the b desired indices or keywords.

A trivial batch PIR scheme is to execute b instances of a keyword/index PIR scheme in parallel. Doing so will require b times the server computation cost of the underlying keyword/index PIR scheme. Batch PIR schemes in the literature aim to make server computation depend only on N and not on the batch size b. The communication cost inevitably depends on b since the client needs to receive the b entries.

Note that preprocessing-based PIR schemes also assume that the client wants to retrieve many entries from the database. So, preprocessing-based PIR and batch PIR are two paradigms for improving the *amortized* server computation of PIR. The difference between the two is that a batch PIR requires the client to have the entire batch ready upfront, whereas a preprocessing-based PIR allows the client to generate the sequence of queries one by one, potentially dependent on the answers to prior queries. In this sense, preprocessing-based PIR is a stronger primitive and, as expected, is harder to construct.

B. Batch PIR using a Cuckoo Hash Table

We describe the batch PIR scheme from Angel et al. [5], [31]. For concreteness, we describe the keyword version of batch PIR. The index version is similar. The construction is given in Algorithm 10.

The construction uses a cuckoo hash table with three hash functions (see Section VIII-B). The client constructs a cuckoo hash table CH by inserting all b keywords. Recall that the cuckoo hash table with three hash functions should have size $|\mathsf{CH}| \approx 1.5b$. Let m denote $|\mathsf{CH}|$.

The server creates m *buckets*. The server then adds each entry in the database to *all three* buckets specified by the cuckoo hash functions $\mathsf{h}_1, \mathsf{h}_2, \mathsf{h}_3$. To elaborate, using $\mathsf{B}[i]$ to denote the i-th bucket, each entry $(k, v) \in \mathsf{DB}$ is added to buckets $\mathsf{B}[\mathsf{h}_1(k)]$, $\mathsf{B}[\mathsf{h}_2(k)]$, *and* $\mathsf{B}[\mathsf{h}_3(k)]$.

Then, the client and the server execute m *single-query* keyword PIR instances. In the i-th instance, the client's keyword is $\mathsf{CH}[i]$ and the server's input is the i-th bucket $\mathsf{B}[i]$. Note that out of the m positions in CH, $m - b \approx 0.5b$ are empty. For each of these keyword PIR instances, the client can use a dummy keyword to make the query and discard the dummy answer. The answers from the other b keyword PIR instances are returned as the answer of the batch PIR.

Correctness and Privacy. Privacy follows from the privacy of the underlying keyword PIR. For correctness, barring the small failure probability of cuckoo hash table insertions, it suffices to show that if a desired keyword k appears in $\mathsf{CH}[i]$, then the corresponding bucket $\mathsf{B}[i]$ contains the corresponding database entry. This is the case because each entry is added to *all three* buckets corresponding to the three candidate positions in CH in which its keyword can appear.

We remark that, contrary to the situation in keyword PIR, the failure probability of the cuckoo hash table *may* affect the correctness or privacy of batch PIR, depending on the implementation details. The difference arises from the fact that the keyword PIR scheme in Section VIII builds a cuckoo hash table on a set of *public* database entries, whereas here the client builds a cuckoo hash table on its *secret* input. This is why the scheme uses three hash functions to keep the failure probability small.

Efficiency. The total number of entries in the server's buckets combined is $3N$, where N is the number of entries in the database. If we plug in a keyword PIR scheme with linear server computation (Section VIII), then the server computation cost of the batch keyword PIR is linear in N and independent of the batch size b. The communication cost of the batch keyword PIR, on the other hand, is proportional to the number of buckets m, which is $O(b)$.

C. A Brief Survey of Batch PIR

Batch PIR was first studied by Beimel et al. [57]. They slightly reduced server computation for a batch of queries (possibly from different clients) using fast matrix multiplication. A line of work [87], [88] uses batch codes to obtain batch PIR from single-query PIR. This approach achieves greater savings in server computation, but the communication is much worse. Also, the batch of queries must come from a single client, a constraint adopted by most subsequent schemes, including the one we presented. The batch PIR scheme we presented in this section is from Angel et al. [5], [31]. Its use of a cuckoo hash table can be viewed as a probabilistic batch code, which has a small chance of failure but gives much better computation and communication costs. There are also customized schemes that are not black-box transformations from single-query PIR [53], [89], [90].

X. Practical Performance Characterization

This section aims to give insights into the practical performance of the different PIR paradigms we reviewed in this article. We start with a disclaimer that it is difficult, if not impossible, to accurately characterize the performance of a wide variety of schemes across different paradigms in a short section. Different paradigms or schemes may choose to prioritize different metrics and exhibit different trade-offs. Some schemes come with many parameters that can be tuned to optimize for different metrics, database sizes, or entry sizes. Discrepancies in security assumptions, intended deployment scenarios, and engineering qualities further complicate the issue. As such, the numbers in this section should be read as rough estimates rather than precise evaluations. Nevertheless, we believe it is helpful for the reader to have a rough idea of how each paradigm of PIR fares in practice.

A. Methodology

As discussed earlier, the main efficiency metrics for PIR are query size, response size, and server computation. For schemes with linear server computation, we follow the literature and measure server computation using *throughput*, defined as the database size (in Bytes) divided by the server computation time (in seconds). Thus, higher throughput indicates faster server computation. The advantage of using throughput is that it remains largely stable when the database size or shape changes

TABLE II
PRACTICAL PERFORMANCE CHARACTERIZATION OF MAJOR PIR PARADIGMS.

		Two-Server Information-Theoretic	Two-Server Computational (DPF)	Single-Server Computational	Single-Server Server-preprocessing	Two-Server Stateful	Single-Server Stateful
				1 GB database		(32-byte entries)	
Online	Query Size	23 KB	1 KB	15 KB	390 KB	0.5 KB	120 KB
	Response Size	23 KB	2×	11 KB or 4×	12 KB or 3×	4 KB	120 KB
	Server Computation	16 GB/s	15 GB/s	1 GB/s	3.5 GB/s	–	13.5 GB/s
Offline	Communication			1.5 MB		16 MB	120 MB
	Server Computation			10 s	150 s	20 s	120 s
				16 GB database		(32-byte entries)	
Online	Query Size	90 KB	1 KB	15 KB	1060 KB	2 KB	480 KB
	Response Size	90 KB	2×	11 KB or 4×	12 KB or 3×	16 KB	480 KB
	Server Computation	16 GB/s	15 GB/s	1.4 GB/s	6.6 GB/s	–	13.5 GB/s
Offline	Communication			1.5 MB		64 MB	480 MB
	Server Computation			160 s	2100 s	320 s	2000 s
Cryptographic Primitives		none	PRF	RLWE SHE	LWE & RLWE SHE	PRF	LWE SHE

– The efficiencies of the first four schemes depend primarily on the database size. They are largely independent of the database's entry size, at least for common entry sizes ranging from a few bytes to a few kilobytes, so we do not specify an entry size for these four schemes.
– The efficiencies of the two stateful schemes are sensitive to both the database size and the entry size. We use 32-byte entries for them as examples.
– An empty cell indicates that the metric is not applicable for the paradigm. A dash (–) indicates that the cost is too small to be worth reporting.
– Offline communication in single-server computational PIR is from the client to the server, and the server stores data of that size *per client*.
– Offline communication in stateful PIRs is from the server to each client, and each client stores hints about that size.

(as long as the database is not too small). For a scheme with sublinear server computation, the server computation is insignificant and not worth reporting. For all but one scheme, the client computation is insignificant and not worth reporting, so we omit client computation from the table. For schemes that use preprocessing, we report the computation and communication for the online and offline phases separately.

We report the numbers using a combination of back-of-the-envelope calculations and experimental evaluation using available implementations. Most PIR schemes are embarrassingly parallel. Thus, for simplicity, we use a single CPU core for both the client and the server.

B. Performance Characterization

Table II provides a performance characterization of the mainstream PIR paradigms reviewed in this article.

Two-server information-theoretic PIR. We use the variant of Algorithm 2 that balances query size and response size. The communication cost is the square root of the database size in bytes. The server computation throughput is very high since the only server-side computation (beyond reading the database) is bitwise XOR.

Two-server computational PIR. We use the DPF-based scheme in Algorithm 3 and its implementation at https://github.com/dkales/dpf-cpp. The DPF-based PIR achieves a good balance between communication and computation. The query (the DPF keys) is around 1 KB. The response size is only $2\times$ of the plaintext entry size. The server computation throughput is slightly lower than that of the information-theoretic scheme.

Single-server computational PIR. We use the updated OnionPIR [56] and its implementation at https://github.com/chenyue42/OnionPIRv2. Due to the use of RLWE-based SHE, the server computation is roughly an order of magnitude more expensive (lower throughput) than the two-server schemes. The query size and the response size are both at least tens of kilobytes, as even a single lattice ciphertext is about that size. If a database entry does not fit in one ciphertext, the response size is about $4\times$ of the plaintext entry size. The default mode of the scheme requires the client to send, and the server to store, additional helper data for each client, resulting in offline communication and extra server storage. Alternatively, the client can send the helper data as part of each query, incurring that cost online while avoiding offline communication and per-client server storage.

Single-server server-preprocessing PIR. We use InsPIRe [62] and its implementation at https://github.com/google/private-membership/tree/main/research/InsPIRe. InsPIRe leverages LWE-based SHE and server preprocessing to improve online server computation over RLWE-based single-server PIR schemes. InsPIRe can be parameterized to achieve different trade-offs between query size and online server computation. We pick a parameter setting that gives noticeable improvements in online server computation over RLWE without excessively increasing the query size. The response size of InsPIRe is comparable to RLWE-based single-server PIR schemes. The main downsides of the scheme are large query size and inefficiency in handling database updates. (At the time of writing, a bug in the InsPIRe code results in large client computation, but it is easily fixable.)

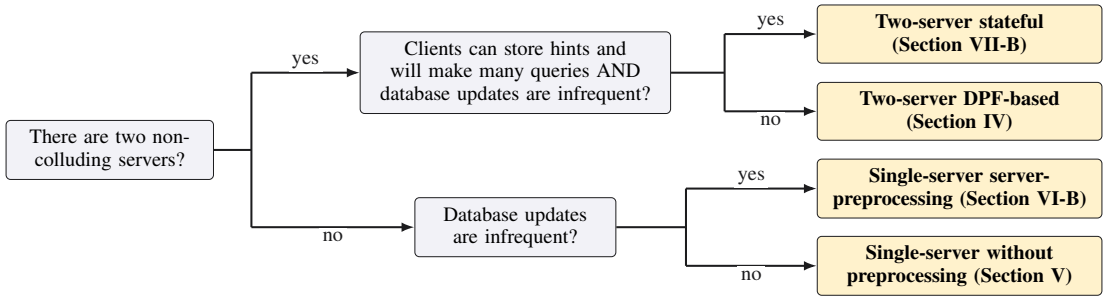


Fig. 8. Guidelines for selecting a PIR paradigm given application and deployment constraints.

Two-server stateful PIR. We use the two-server variant of HarmonyPIR [91] and its implementation at https://github.com/PranavShriram/HarmonyPIR. Two-server stateful PIR can make the server computation square root in the database size, and hence insignificant. The query size and response size are also square root in the size of the database. The main downsides are the need for client storage and offline server computation. Many sublinear stateful PIR schemes, including HarmonyPIR, can trade off client storage for amortized communication and computation costs in a proportional manner. The data reported in the table represents an all-around efficient (but nevertheless arbitrary) parameter choice.

Single-server stateful PIR. Although there are single-server stateful schemes with sublinear server computation [76], [77], [79], [91], [92], these schemes require the client to stream the entire database during the offline phase. Thus, we use SimplePIR [67], a single-server stateful scheme with linear server computation, and its implementation at https://github.com/ahenzinger/simplepir. SimplePIR uses LWE with preprocessing (see Section VI-B). As a result, its online server computation throughput is better than that of single-server stateless schemes and close to that of two-server stateless schemes. But it has to make significant sacrifices on other metrics, including high communication (both offline and online) and large client storage. The original SimplePIR implementation requires substantial server preprocessing and online client computation, but these can be significantly reduced using a structured matrix $\mathbf{A}$ and RLWE techniques (see Section VI-B and [93]).

C. Deployment Guidelines

Given the performance characteristics of different PIR paradigms summarized in Table II, we provide high-level guidelines in Figure 8 for selecting an appropriate PIR paradigm in practice based on the nature of the application and deployment constraints. These guidelines account for the most important factors and most common situations, but they are not intended to be exhaustive. Specific applications may have unique requirements or constraints that justify alternative design choices.

When it is reasonable to assume the existence of two non-colluding servers, two-server PIR paradigms are generally preferred because they tend to be substantially more efficient than single-server paradigms. In particular, the two-server computational PIR based on DPF (Section IV) is about as efficient as a stateless PIR scheme can be. Two-server stateful PIR (Section VII-B) can further reduce server computation to amortized sublinear, but it comes with several requirements. First, clients must be able to store hints. Second, clients must plan to issue many queries over time to effectively amortize the high cost of the offline phase. Third, database updates must be infrequent, since each update requires additional work to refresh client hints. When all three conditions are satisfied, two-server stateful PIR may be preferred; otherwise, two-server DPF-based PIR is recommended.

The assumption of two non-colluding servers, however, may not always be convincing. In practice, the two servers must coordinate closely to maintain identical and up-to-date copies of the database. Therefore, even when the two servers are operated by independent and reputable organizations, assuming they never collude—whether intentionally or inadvertently—may be difficult to justify. Single-server PIR paradigms avoid the non-collusion assumption, albeit at higher costs.

With a single server, the server-preprocessing paradigm (Section VI-B) can improve online server computation (and possibly communication) by constant factors. However, if the database is frequently updated, the expensive preprocessing phase must be rerun frequently, likely negating the benefits. Consequently, single-server PIR with server-preprocessing (Section VI-B) is recommended for static or rarely updated databases, while single-server PIR without preprocessing (Section V) is more suitable for frequently updated databases.

We omit two-server information-theoretic PIR (Section III) and single-server stateful PIR (Section VII) from the above discussion because, in our view, they are not the most competitive options in most settings in practice.

XI. Sample Applications

A. Live Caller ID Lookup

In 2024, Apple added a new feature, Live Caller ID, to its phones [2]. A user's phone will display the identity of the incoming caller even if the caller is not in the user's contact list, or display "spam likely" if the caller has been reported by other users as such. To support this feature, Apple can maintain a database to map phone numbers to identities ("suspected spam" can be a special identity). When a user receives a call, the user's phone could look up the database using the incoming caller's number. However, this would allow the server to learn who is calling the user, which is sensitive

information that the user may not wish to disclose to Apple. To enable the Live Caller ID feature without revealing the caller's number to Apple, the user's phone sends a keyword PIR query (the keyword is the caller's number) to retrieve the identity associated with the caller's number [2].

B. URL Filtering

Another application of PIR deployed by Apple is URL filtering [2]. In this case, a browser or a mobile app (especially one in parental control mode or designed for children) may want to block certain websites that are known to be harmful or malicious. Similarly, Apple (or another company) can maintain a database that maps URLs to website categories. But again, the user may not want to reveal to the Apple server which URL (legitimate or otherwise) it is visiting. PIR enables this feature without revealing the user's browsing history to the server [2]. The browser/app can send a keyword PIR query (the keyword is the URL) to retrieve what type of website the URL corresponds to. The browser/app can then block a harmful URL and display a reason.

C. Password Checkup

Billions of user passwords have been exposed as part of data breaches that routinely make headlines. A user would like to check if his passwords have been exposed and, if so, choose new passwords. Some web services have compiled databases of exposed passwords for users to query. But it is a bad idea for a user to send their plaintext password to query such a database – this is the very act of exposing their password. Keyword PIR is a natural solution here (the keyword is the user's password) by allowing a user to check whether their password is in a database of exposed passwords without revealing their password to the server [6], [7].

D. Anonymous Communication

Anonymous communication protocols allow users to exchange messages without others (such as Internet Service Providers) finding out that they communicated. There are several proposals for anonymous communication systems [3]–[5] that use PIR. We describe a natural approach used in Angel et al. [5].

Consider a system consisting of many users who wish to talk to each other anonymously and an untrusted server to facilitate. The protocol proceeds in rounds. In each round, each user sends a key-value pair to the server, where the key is the output of a PRF and the value is a message encrypted for its intended recipient. Then, each user retrieves the encrypted message intended for it using a single-server keyword PIR. For this to work, the two users communicating need to agree on the PRF secret key. Readers can refer to [5] for more details. It is not hard to see that the server cannot link the sender and recipient of a message due to PIR.

PIR has also been suggested [3] to improve the widely used anonymous communication system Tor [94]. At a high level, Tor consists of thousands of volunteer relays. Each user randomly selects several relays to route their network traffic. Each relay sees only the previous and next hops, so the destination of the user's traffic remains hidden. For this to work, the user needs to retrieve the public keys and IP addresses of the relays from a Tor *directory authority*. However, if the user only downloads information for the relays she uses, then the directory authority can monitor the traffic of these relays and identify who she is communicating with. Currently, each user downloads the information for *all* relays (essentially a trivial PIR). Mittal et al. [3] suggest that (index) PIR can be used to retrieve information about the selected relays (such as their public keys and IP addresses) without revealing which relays are selected.

XII. Conclusion

Private Information Retrieval (PIR) provides a principled cryptographic approach for users to retrieve data from public databases while keeping their queries hidden from service providers and network observers. In this article, we surveyed the main design paradigms for PIR by organizing the design space along three key aspects: the number of servers, the security assumptions, and the use of preprocessing. We present conceptually simple constructions to convey the core ideas and essential techniques underlying each paradigm. We provided an overview of the performance characteristics of different PIR paradigms and their trade-offs. We also described common applications of PIR and discussed important extensions of PIR, including keyword PIR and batch PIR, which are crucial for practical deployments where data is usually indexed by arbitrary strings and multiple queries naturally occur.

PIR remains an active area of research. Further improving performance (both asymptotically and concretely), relaxing assumptions, supporting dynamic databases, developing better implementations, and deploying real applications all continue to be interesting and important research directions. We hope this article equips readers with a clear conceptual map of the PIR landscape and serves as a starting point for further exploration of both the theoretical foundations and practical systems of Private Information Retrieval.

Acknowledgments

We thank Srinivas Devadas, David Heath, David Wu, Arthur Lazzaretti, Joon Seo, Kevin Yeo, Ali Atiia, Benny Pinkas, Kabir Tomer, and anonymous reviewers for their insightful feedback on the article. This work is funded in part by the National Science Foundation award #2246386 and gifts from Google.

References

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

[2] "Combining machine learning and homomorphic encryption in the apple ecosystem," https://machinelearning.apple.com/research/homomorphic-encryption.

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

[4] A. Kwon, D. Lazar, S. Devadas, and B. Ford, "Riffle: An efficient communication system with strong anonymity," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 2, pp. 115–134, 2015.

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

[6] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh, and E. Bursztein, "Protecting accounts from credential stuffing with password breach alerting," in *28th USENIX Security Symposium*, 2019, pp. 1556–1571.

[7] A. Ali, T. Lepoint, S. Patel, M. Raykova, P. Schoppmann, K. Seth, and K. Yeo, "Communication–computation trade-offs in PIR," in *30th USENIX Security Symposium*, 2021, pp. 1811–1828.

[8] A. Juels, "Targeted advertising... and privacy too," in *Cryptographers' Track at the RSA Conference*. Springer, 2001, pp. 408–424.

[9] M. Backes, A. Kate, M. Maffei, and K. Pecina, "Obliviad: Provably secure and practical online behavioral advertising," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 257–271.

[10] M. Green, W. Ladd, and I. Miers, "A protocol for privately reporting ad impressions at scale," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1591–1601.

[11] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish, "Scalable and private media consumption with popcorn," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp. 91–107.

[12] A. Beimel, Y. Ishai, E. Kushilevitz, and T. Malkin, "One-way functions are essential for single-server private information retrieval," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, 1999, pp. 89–98.

[13] G. Di Crescenzo, T. Malkin, and R. Ostrovsky, "Single database private information retrieval implies oblivious transfer," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 122–138.

[14] Y. Ishai, E. Kushilevitz, and R. Ostrovsky, "Sufficient conditions for collision-resistant hashing," in *Theory of Cryptography Conference*. Springer, 2005, pp. 445–456.

[15] R. Impagliazzo and S. Rudich, "Limits on the provable consequences of one-way permutations," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, 1989, pp. 44–61.

[16] D. R. Simon, "Finding collisions on a one-way street: Can secure hash functions be based on general assumptions?" in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1998, pp. 334–345.

[17] M. Rabin, "How to exchange secrets by oblivious transfer," *Technical Memo TR-81*, 1981.

[18] C. Meadows, "A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party," in *1986 IEEE Symposium on Security and Privacy*. IEEE, 1986, pp. 134–134.

[19] B. A. Huberman, M. Franklin, and T. Hogg, "Enhancing privacy and trust in electronic communities," in *Proceedings of the 1st ACM conference on Electronic commerce*, 1999, pp. 78–86.

[20] C. Lin, Z. Liu, P. Miao, and M. Tromanhauser, "Finding balance in unbalanced PSI: A new construction from single-server PIR," *IACR Communications in Cryptology*, vol. 2, no. 1, 2025.

[21] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 431–473, 1996.

[22] E. Stefanov, M. v. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," *Journal of the ACM (JACM)*, vol. 65, no. 4, pp. 1–26, 2018.

[23] Z. Dvir and S. Gopi, "2-server PIR with subpolynomial communication," *Journal of the ACM (JACM)*, vol. 63, no. 4, pp. 1–15, 2016.

[24] K. Efremenko, "3-query locally decodable codes of subexponential length," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 39–44.

[25] S. Yekhanin, "Towards 3-query locally decodable codes of subexponential length," *Journal of the ACM (JACM)*, vol. 55, no. 1, pp. 1–16, 2008.

[26] A. Beimel, Y. Ishai, E. Kushilevitz, and I. Orlov, "Share conversion and private information retrieval," in *2012 IEEE 27th Conference on Computational Complexity*. IEEE, 2012, pp. 258–268.

[27] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond, "Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.* IEEE, 2002, pp. 261–270.

[28] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer, 2014, pp. 640–658.

[29] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1292–1303.

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

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

[32] M. H. Mughees, H. Chen, and L. Ren, "OnionPIR: Response efficient single-server PIR," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2292–2306.

[33] S. J. Menon and D. J. Wu, "Spiral: Fast, high-rate single-server PIR via FHE composition," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 930–947.

[34] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, Sep. 2009. [Online]. Available: https://doi.org/10.1145/1568318.1568324

[35] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *J. ACM*, vol. 60, no. 6, Nov. 2013. [Online]. Available: https://doi.org/10.1145/2535925

[36] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology – CRYPTO 2012*. Berlin, Heidelberg: Springer, 2012, pp. 868–886.

[37] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," IACR Cryptology ePrint Archive, Paper 2012/144, 2012. [Online]. Available: https://eprint.iacr.org/2012/144

[38] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. New York, NY, USA: ACM, 2012, p. 309–325. [Online]. Available: https://doi.org/10.1145/2090236.2090262

[39] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT 2017*. Springer, 2017, pp. 409–437.

[40] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 2009, p. 169–178. [Online]. Available: https://doi.org/10.1145/1536414.1536440

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

[42] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, pp. 270–299, 1984.

[43] J. P. Stern, "A new and efficient all-or-nothing disclosure of secrets protocol," in *Advances in Cryptology — ASIACRYPT'98*. Berlin, Heidelberg: Springer, 1998, pp. 357–371.

[44] Y.-C. Chang, "Single database private information retrieval with logarithmic communication," in *Information Security and Privacy*. Berlin, Heidelberg: Springer, 2004, pp. 50–61.

[45] H. Lipmaa, "An oblivious transfer protocol with log-squared communication," in *Information Security*. Berlin, Heidelberg: Springer, 2005, pp. 314–328.

[46] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system," in *Public Key Cryptography*. Berlin, Heidelberg: Springer, 2001, pp. 119–136.

[47] C. Cachin, S. Micali, and M. Stadler, "Computationally private information retrieval with polylogarithmic communication," in *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*. Springer, 1999, pp. 402–414.

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

[49] R. Ostrovsky and W. E. Skeith, "A survey of single-database private information retrieval: techniques and applications," in *Proceedings of the 10th International Conference on Practice and Theory in Public-Key Cryptography*. Berlin, Heidelberg: Springer-Verlag, 2007, p. 393–411.

[50] C. Gentry and S. Halevi, "Compressible FHE with applications to PIR," in *Theory of Cryptography*. Springer, 2019, pp. 438–464.

[51] J. Park and M. Tibouchi, "SHECS-PIR: somewhat homomorphic encryption-based compact and scalable private information retrieval," in *European symposium on research in computer security*. Springer, 2020, pp. 86–106.

[52] M. Luo, F.-H. Liu, and H. Wang, "Faster FHE-based single-server private information retrieval," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 1405–1419.

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

[54] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, 2011, pp. 97–106.

[55] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, p. 34–91, Jan. 2020. [Online]. Available: https://doi.org/10.1007/s00145-019-09319-x

[56] Y. Chen and L. Ren, "OnionPIRv2: Efficient single-server PIR," Cryptology ePrint Archive, Paper 2025/1142, 2025. [Online]. Available: https://eprint.iacr.org/2025/1142

[57] A. Beimel, Y. Ishai, and T. Malkin, "Reducing the servers computation in private information retrieval: PIR with preprocessing," in *20th Annual International Cryptology Conference (CRYPTO)*. Springer, 2000, pp. 55–73.

[58] W.-K. Lin, E. Mook, and D. Wichs, "Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE," in *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, 2023, pp. 595–608.

[59] A. Henzinger, E. Dauterman, H. Corrigan-Gibbs, and N. Zeldovich, "Private web search with tiptoe," in *Proceedings of the 29th Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2023, p. 396–416. [Online]. Available: https://doi.org/10.1145/3600006.3613134

[60] B. Li, D. Micciancio, M. Raykova, and M. Schultz-Wu, "Hintless single-server private information retrieval," in *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference*. Berlin, Heidelberg: Springer-Verlag, 2024, p. 183–217. [Online]. Available: https://doi.org/10.1007/978-3-031-68400-5_6

[61] S. J. Menon and D. J. Wu, "YPIR: High-Throughput Single-Server PIR with silent preprocessing," in *33rd USENIX Security Symposium*. USENIX Association, Aug. 2024, pp. 5985–6002.

[62] R. A. Mahdavi, S. Patel, J. Y. Seo, and K. Yeo, " InsPIRe: Communication-Efficient PIR with Server-side Preprocessing ," in *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, May 2026, pp. 1240–1259. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP63933.2026.00061

[63] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (ring) LWE ciphertexts." Berlin, Heidelberg: Springer-Verlag, 2021, p. 460–479. [Online]. Available: https://doi.org/10.1007/978-3-030-78372-3_18

[64] H. Okada, R. Player, S. Pohmann, and C. Weinert, "Towards practical doubly-efficient private information retrieval," in *International Conference on Financial Cryptography and Data Security*. Springer, 2024, pp. 264–282.

[65] E. Boyle, Y. Ishai, R. Pass, and M. Wootters, "Can we access a database both locally and privately?" in *Theory of Cryptography: 15th International Conference*. Springer, 2017, pp. 662–693.

[66] R. Canetti, J. Holmgren, and S. Richelson, "Towards doubly efficient private information retrieval," in *Theory of Cryptography: 15th International Conference*. Springer, 2017, pp. 694–726.

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

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

[69] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan, "Single-server private information retrieval with sublinear amortized time," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 3–33.

[70] B. Fisch, A. Lazzaretti, Z. Liu, and C. Papamanthou, "ThorPIR: single server PIR via homomorphic thorp shuffles," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 1448–1462.

[71] Y. Ishai, E. Shi, and D. Wichs, "PIR with client-side preprocessing: information-theoretic constructions and lower bounds," in *Annual International Cryptology Conference*. Springer, 2024, pp. 148–182.

[72] S. Patel, G. Persiano, and K. Yeo, "Private stateful information retrieval," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1002–1019.

[73] H. Corrigan-Gibbs and D. Kogan, "Private information retrieval with sublinear online time," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2020, pp. 44–75.

[74] K. Yeo, "Lower bounds for (batch) PIR with private preprocessing," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 518–550.

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

[76] M. Zhou, A. Park, W. Zheng, and E. Shi, "Piano: Extremely simple, single-server PIR with sublinear server computation," in *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2024.

[77] L. Ren, M. H. Mughees, and I. Sun, "Simple and practical amortized sublinear private information retrieval using dummy subsets," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 1420–1433.

[78] A. Lazzaretti and C. Papamanthou, "Single pass client-preprocessing private information retrieval," in *33rd USENIX Security Symposium*, 2024, pp. 5967–5984.

[79] Z. Wang and L. Ren, "Single-server client preprocessing PIR with tight space-time trade-off," in *44th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Berlin, Heidelberg: Springer-Verlag, 2025, p. 94–122. [Online]. Available: https://doi.org/10.1007/978-3-031-91095-1_4

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

[81] M. Zhou, W.-K. Lin, Y. Tselekounis, and E. Shi, "Optimal single-server private information retrieval," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 395–425.

[82] A. Lazzaretti and C. Papamanthou, "Near-optimal private information retrieval with preprocessing," in *Theory of Cryptography Conference*. Springer, 2023, pp. 406–435.

[83] S. Patel, J. Y. Seo, and K. Yeo, "Don't be dense: efficient keyword PIR for sparse databases," in *32nd USENIX Security Symposium*, 2023, pp. 3853–3870.

[84] B. Chor, N. Gilboa, and M. Naor, *Private information retrieval by keywords*. Technion-IIT, Department of Computer Science, 1997.

[85] S. Celi and A. Davidson, "Call me by my name: Simple, practical private information retrieval for keyword queries," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4107–4121.

[86] M. Hao, W. Liu, L. Peng, C. Zhang, P. Wu, L. Zhang, H. Li, and R. H. Deng, "Practical keyword private information retrieval from key-to-index mappings," in *Proceedings of the 34th USENIX Conference on Security Symposium*. USENIX Association, 2025.

[87] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Batch codes and their applications," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, 2004, pp. 262–271.

[88] R. Henry, "Polynomial batch codes for efficient IT-PIR." in *PoPETs 2016*, 2016, pp. 202–218.

[89] M. H. Mughees and L. Ren, "Vectorized batch private information retrieval," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 437–452.

[90] A. Bienstock, S. Patel, J. Y. Seo, and K. Yeo, "Batch PIR and labeled PSI with oblivious ciphertext compression," in *33rd USENIX Security Symposium*, 2024, pp. 5949–5966.

[91] P. S. Arunachalaramanan and L. Ren, "Efficient single-server stateful pir using format-preserving encryption," *Cryptology ePrint Archive*, 2026.

[92] ——, "Single-server stateful PIR with verifiability and balanced efficiency," in *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2026, to appear.

[93] R. Lehmkuhl, A. Henzinger, and H. Corrigan-Gibbs, "Distributional private information retrieval," in *Proceedings of the 34th USENIX Conference on Security Symposium*. USENIX Association, 2025.

[94] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *13th USENIX Security Symposium*. USENIX Association, 2004.