Faster FHE-Based Single-Server Private Information Retrieval

Ming Luo
Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
luoming@iie.ac.cn

Feng-Hao Liu
Washington State University
Pullman, USA
feng-hao.liu@wsu.edu

Han Wang*
Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
wanghan@iie.ac.cn

ABSTRACT

This work introduces KsPIR, a new practically efficient single-server private information retrieval (PIR) system that outperforms the state-of-the-art Spiral (Menon and Wu, S&P 2022) in terms of server response times. We achieve this by proposing novel dimension folding methods, inspired by recent advancements in fully homomorphic encryption. Our methods offer two significant advantages: firstly, they feature simpler designs that eliminate the need for ciphertext expansion steps in Spiral. Secondly, and more importantly, we propose two types of designs that offer distinct advantages - the first type enables preprocessing of the most resource-intensive computation in the offline stage before receiving the query, thereby optimizing online response time; the second type optimizes overall response time without requiring preprocessing in the offline stage, accomplished through a highly optimized baby-step-giant-step matrix-vector homomorphic multiplication.

We conduct comprehensive experiments to evaluate the concrete performance of KsPIR, and the results confirm an approximately 10.7 times faster online throughput than that of Spiral for the first type, and 5.8 times faster overall throughput for the second type.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**.

KEYWORDS

Private Information Retrieval, Fully Homomorphic Encryption

ACM Reference Format:
Ming Luo, Feng-Hao Liu, and Han Wang. 2024. Faster FHE-Based Single-Server Private Information Retrieval . In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3690233

1 INTRODUCTION

Private information retrieval (PIR) enables a client to retrieve a specific element from a server's database without disclosing the index that was queried. This concept was first introduced by Chor et al. [16], and has been widely applied in various domains, including but not limited to anonymous messaging [4, 5, 38, 51], private contact tracing [61], safe browsing [32, 36], and more [24, 62]. The literature on PIR research includes many variations, such as multi-server [18, 22, 36] versus single-server settings [1, 2, 4, 17, 26, 32, 37, 46, 52, 65], and pre-processing (i.e., offline/online) [12, 17–19, 32, 36, 55, 60, 64, 65] versus the plain settings [1, 2, 4, 26, 45, 46, 52, 53]. There has been active research in both theory and practice [12, 31, 39, 40, 56, 58, 60, 63, 64].

Practical Single-Server PIR. This work focuses on the setting of practical single-server PIR. In recent years, several works [1, 2, 4, 45, 46, 52, 54] have focused on implementing and optimizing the concrete performance of single-server PIR. Many of the designs have involved leveraging the power of pre-processing, where the server and client can perform some offline computation before the online query from the client. These works have significantly improved the response time, server's throughputs, and/or the server-client communication complexity, making substantial progress towards practical deployments of PIR in the real-world applications.

In the following, we outline the best practical solutions currently available (to our knowledge) in two general categories. However, solutions in these two categories are incomparable due to various tradeoffs, and each solution has its own strengths and weaknesses. Therefore, depending on the specific requirements of a particular application, users will make distinct selections based on the best fit.

Two General Categories. Below we present two major categories of practical PIR, achieving incomparable advantages in different aspects. The categorization is based on several critical features that PIR aims to optimize. We present some contexts and then the features below. Briefly, an offline/online PIR scheme consists of several phases: (1) There is an offline phase before the client receives the input of the query. In this phase, the server and client can do some one-time set up and pre-process queries (if they have an input-independent component). The server receives the database in this phase. (2) Once the client receives the input of the query, the protocol enters the online phase. Now we present several important features below.

- For the offline phase (including one-time setup and pre-processing offline queries), we list two critical features – the client's computation time and the communication complexity between the server and the client.

*Corresponding author.

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3690233

- For the online phase, we list two critical features – the query and response size, and the server's response time.
- We list two among other important features, including the client's required (long-term) storage and the complexity (for both parties) to handle database updates.

Based on these, two general categories have been studied in the state of the art designs.

Category I. A key characterization is the optimization of client's computation and communication load in both the offline and online phases. Particularly, in all the stages, the client does not storage any database-dependent hints, and the database update is fully client independent. This objective can be accomplished by utilizing generic Fully Homomorphic Encryption (FHE)-based solutions, and there have been recent advancements in efficient implementations, such as XPIR [45], SealPIR [4], SHECS-PIR [54], FastPIR [1], MulPIR [2], OnionPIR [52], Spiral [46] and recent works including HintlessPIR [41], YPIR [49] and WhisPIR [20]. Among these schemes, the currently most efficient options are Spiral (and its related family) [46], HintlessPIR [41], YPIR [49] and WhisPIR [20].

Category II. The designs within this category utilize the client's resources by allocating more intensive offline computation and communication procedures. Moreover, they also involve storing significantly larger hints from the client's side. By using these, the server's online response time (and thus the online throughput) may significantly outperform that in the other category. Notable practical schemes in this category include [36], FrodoPIR [19], Simple/Double PIR [32] and Piano [65]. The current state-of-the-art schemes are Simple/Double PIR [32] and Piano [65]. However, it is important to note that these two schemes are incomparable due to the tradeoffs in different aspects. In Section 6.4, we will delve into further details.

Upon examining the features, we observe that Category I is better suited for light-weighted clients who require minimal storage for hints, computation, and communication across all stages. Additionally, these clients do not need to repeat extensive computation even when the database undergoes frequent updates (insertions or modifications). On the other hand, Category II becomes more advantageous when clients are actively involved in the offline phase and can store larger hints. In such cases, Category II schemes can provide significantly faster online response times. We also notice that due to the lower bound result of $tr = \Omega(n)$ [7] where t is server online time, r is the bits of hint, and n is the size of database, for schemes in Category I, the server's computation complexity would inherently be linear in the database size, whereas for schemes in Category II, it is possible to design sublinear computation for servers with larger hints stored on the client's side, e.g., [65].

It is worth noting that both categories hold their own significance and may offer different advantages in various scenarios. Thus, advancements in either category hold value for practical deployments. Table 1 presents a concise overview of the pros and cons of these two categories, along with the state-of-the-art schemes and our results figuratively. This would allow for a quick comparison and understanding of the different aspects of each category and our merits.

	Spiral [46]	This paper	Simple/Double PIR [32]	Piano [65]
Offline (Preprocessing) phase				
Client comp.	✔	✔	✔	✓
Comm.	✔	✔	✓	✗
Online phase				
Server response time	✓	✓✓	✔	✔✔
Online comm.	✔	✔	✔	✔
Others				
Client storage	✔	✔	✓	✓
Database update (Client independent)	✔	✔	✗	✗

Table 1: Performance characteristics of the four PIR protocols. "Comm." stands for communication cost and "Comp." stands for computation cost. ✔ means "the best". ✗ means "not so good". ✓ means "the performance is between ✔ and ✗". Two symbols indicate more, i.e., ✔✔ is better than ✔ and ✓✓ is better than ✓.

1.1 Our Results

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

- We develop three efficient PIR schemes in the offline/online setting. All our three schemes are in Category I, as the client stores minimal storage (no database-dependent hints) and the database update is fully client independent.
  Our designs are built upon the dimension folding idea of the Spiral framework [46], yet with several enhancements:
  - Our dimension folding algorithms are simpler than those in [46], eliminating the need for the computationally intensive expansions of ciphertexts.
  - In the first two designs, we identify a crucial advantage: the client's query can be split into an offline part and an online part, where the heaviest computational load lies in processing the offline part. As the offline part does not require knowing the specific index queried, this heavy computation can be done beforehand, significantly reducing the online response time and achieving a substantial improvement in online throughput.
  - In the third design, we encode the database and the client's query differently and deploy the highly optimized "Baby-step-Giant-step" method [30, 34]. This approach does need a slightly longer online time than the first two designs, yet eliminates the need for preprocessing, resulting in better overall throughput (offline + online).

  Our three designs offer incomparable advantages, providing flexible options for application designers to choose from depending on the specific scenarios and requirements.
- We implement our schemes in C++ with library [43] to evaluate the concrete performances. Particularly, our first and second constructions confirm an approximately 10.7× faster online runtime than that of Spiral, and our third construction confirms an

approximately 5.8× faster overall throughput, under the 256 MB database configuration. This demonstrates practicality of our work. More details are presented in Section 6.3 and Section 6.4.

Table 1 presents a figurative comparison between Categories I and II, and our progress in the Category I. The schemes in the table represent the best practical solutions in the categories. By looking at this table, the readers can quickly get the general differences between these categories, and our improvements in the first category.

1.2 Technical Overview

Briefly, our approach is the classic FHE-based construction where the client sends an encryption of the position (denoted by Enc(index)) as the query to the server. The server who holds the database DB homomorphically computes Enc(DB[index]) and returns the resulting ciphertext to the client. The client can decrypt and retrieve the answer. This approach has been implemented in prior works [1, 2, 4, 26, 45, 46, 52, 53], yet how to optimize the concrete efficiency remains the main research challenge. Particularly, it is important to determine the most suitable FHE schemes/variants, parameters, and the homomorphic methods to achieve competitive performances.

As our designs use different encodings for the database and queries, we would first present the encoding and then our insights.

Encoding Method for the First Two Constructions. We encode the database into a two dimensional array, say $\mathsf{DB} \in \mathbb{Z}_p^{n\times n}$, and use polynomial $t_i(X)$ of degree n to encode the i-th column into its coefficients, for $i \in [n]$. To query the database of index $= (u, w) \in [n] \times [n]$, our schemes use RLWE [44, 59] and RGSW [3, 15, 21, 28] schemes for the two dimensions, respectively, i.e., $\mathsf{RLWE.Enc}(X^{-u})$ and $\mathsf{RGSW.Enc}(X^{-w})$. To achieve more efficient homomorphic responses, we develop critical techniques from the insights below.

Insight of First Construction. We identify several nice properties of RLWE and RGSW as developed along the research of FHE:

(1) Given $\mathsf{RLWE.Enc}(X^{-u})$ and $t_i(X)$, we can efficiently compute and extract an LWE ciphertext that encrypts $t_i[u]$, the coefficient with respect to X^u of $t_i(X)$. This has been used in [15, 21] and many follow up works. In the end, we have n LWE ciphertexts.
(2) Given n LWE ciphertexts that encrypt $t_0[u], \ldots, t_{n-1}[u]$, we can convert them into a RLWE ciphertext that encrypts $t(X) = t_0[u] + t_1[u]X + \cdots + t_{n-1}[u]X^{n-1}$. This can be achieved by using the key-switching technique of [13, 50].
(3) Given ciphertexts $\mathsf{RGSW.Enc}(X^{-w})$ and $\mathsf{RLWE.Enc}(t(X))$, we can compute the *external product* [15], resulting in a RLWE ciphertext that encrypts $t_w[u]$ in the constant term. Then one can extract an LWE ciphertext that encrypts $t_w[u]$. This is the resulting response, where the client can retrieve the answer by decrypting it.

In Figure 1, we present a diagram for the above process. We notice that the first two steps are referred to as "first dimension folding" and the last step as "second dimension folding", where the names represent their behavior. The dimension folding technique is conceptually similar to that in Spiral [46], yet our instantiation can avoid the heavy ciphertext expansions as used by Spiral [46]. This already gives a non-trivial advantage of our design.

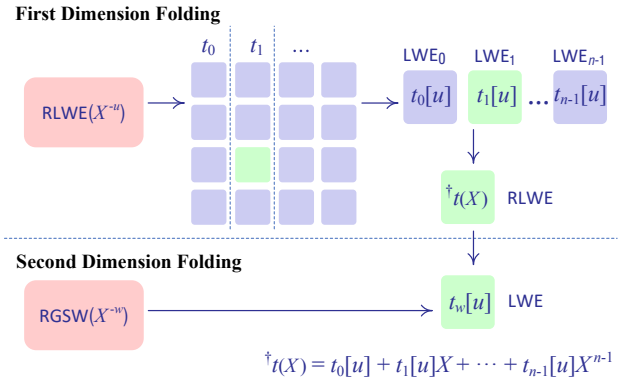

Figure 1: A basic PIR protocol in two dimensions. The query consists of a RLWE ciphertext and a RGSW ciphertext. The response is an LWE ciphertext.

Insight of Second Construction. Our major technique to substantially improve the online throughput relies on the second insight as we now present. We first observe that the above steps (1) and (2) are more computationally heavy, whereas step (3) is light. Nevertheless, the majority burdens in steps (1) and (2) can be done in an offline manner, which is the key that substantially improves the online throughput. Next we highlight the ideas.

First we observe that each RLWE ciphertext consists of two ring elements, say (b, a), where a is uniformly random and thus message independent, and only b is message dependent. In the end Step (1), we would obtain n LWE ciphertexts $(b_0, \boldsymbol{a}_0), \cdots, (b_{n-1}, \boldsymbol{a}_{n-1})$. (Here the readers can understand the high level ideas without going into the precise space of LWE ciphertexts.) We observe that $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{n-1}$ can be derived from a and the database DB, without knowing b. Thus, this step can be computed in an offline manner.

For step (2), we can use the conversion algorithm of [50] – given n LWE ciphertexts $(b_0, \boldsymbol{a}_0), \cdots, (b_{n-1}, \boldsymbol{a}_{n-1})$, the algorithm outputs a RLWE ciphertext (b', a'). Importantly, we identify that this step can also be made into an offline/online manner. Particularly, we only need $\boldsymbol{a}_0, \cdots, \boldsymbol{a}_{n-1}$ to derive a'. Combining with the idea above, we can compute a' in an offline manner. Then in the online stage, one can compute b' from b and the pre-computed a' in a much faster way. As computing a' is the most computationally heavy step, the online step would be much light-weighted and thus the response time can be significantly reduced.

Overall, given a partial RLWE ciphertext a, we can compute a partial RLWE ciphertext a' in an offline manner. Then in the online stage when b is given, the server can then complete the computation of b', and then use (b', a') to proceed to the third step. This would give a substantial improvement of the online efficiency! We present more details of the offline/online technique in Sections 3 and 4.

We notice that our first two schemes have roughly similar overall (offline + online) complexity as the state-of-the-art Spiral [46], yet our designs can push the heaviest computation to the offline stage, resulting in the substantial improvements in the online throughputs.

To improve the overall runtime, we present our third method below.

Encoding of Third Construction. In this construction, we encode the database DB and query index (u, w) as: (1) apply NTT operation to each column of DB to get matrix $\mathbf{M}$; (2) encrypt u and w by RLWE and RGSW to get ciphertexts of $\mathsf{RLWE.Enc}(\mathsf{NTT}^{-1}(\boldsymbol{u}))$ and $\mathsf{RGSW.Enc}(X^{-w})$, respectively, where $\boldsymbol{u}$ is the vector with only one non-zero entry 1 in the u-th position.

Insight of Third Construction. In the first dimension folding, we take $\mathbf{M}$ and $\mathsf{RLWE.Enc}(\mathsf{NTT}^{-1}(\boldsymbol{u}))$ as the input of the "Baby-step-Giant-step" [30, 34], which outputs a RLWE cihpertext encrypting $\mathsf{NTT}^{-1}(\mathbf{M} \cdot \boldsymbol{u})$. Then by linear algebra we have

$$\mathsf{NTT}^{-1}(\mathbf{M} \cdot \boldsymbol{u}) := \mathsf{NTT}^{-1}\big(\mathsf{NTT}(\mathsf{DB}) \cdot \boldsymbol{u}\big) = \mathsf{NTT}^{-1}\big(\mathsf{NTT}(\mathsf{DB} \cdot \boldsymbol{u})\big).$$

Next we can perform the second dimension folding as our first two schemes.

By realizing the above design principle with a highly optimized BSGS implementation, we can significantly improve the overall efficiency, as in this PIR protocol the computation cost (except plaintext-ciphertext multiplication) is rather small. The details and concrete performance analyses are showed in Sections 5 and Section 6.

1.3 Concurrent works

There are several important concurrent and independent works, including HintlessPIR [41], YPIR [49], and WhisPIR [20], each achieving advantages in different aspects. Below, we present a summary of the pros and cons of these concurrent works in some critical features, along with a high-level comparison with our work and the prior state-of-the-art Spiral. It is important to note that each of these schemes offers unique advantages. As PIR is an essential privacy-preserving technology, it is critical to have various options available for users to determine the best tradeoff based on their specific scenarios.

Comparison by Results. We first notice that all these schemes are in Category I and they require very small or even no hint for the clients. Next, we describe some tradeoffs of these schemes and a comparison with ours.

Spiral [46].

- *Pros: Good throughput, large record size (>100 KB) and high rate, where rate = plaintext size / response ciphertext size.*
- *Comparison with ours: We inherit the advantages of Spiral but with approximately 5.8× better throughput. Our query size is slightly larger, i.e., 140 KB, but this is not a significant issue as query size does not affect rate.*

HintlessPIR [41].

- *Pros: High throughput for large databases.*
- *Cons: Rate is not good.*
- *Comparison with ours: Our throughput is better when the database is smaller than 1 GB and our rate is significantly better. When the database reaches 8 GB, their throughput is faster than ours.*

YPIR [49].

- *Pros: Highest throughput.*
- *Cons: The record size is quite smaller.*
- *Comparison with ours: YPIR is limited in many application scenarios. YPIR + Simple PIR [49] can support large records, but its throughput is not as good as YPIR's, and their client-side computational load is greater than that of our protocol.*

WhisPIR [20].

- *Pros: Good throughput and communication.*
- *Comparison with ours: Our throughput and rate are slightly better than WhisPIR.*

Comparison by Techniques. We notice that there is a common technique shared by HintlessPIR [41], YPIR [49] and WhisPIR [20]. Briefly, clients upload key-switching keys during the online phase, thereby eliminating the need for the server to maintain any client-specific storage. This also allows certain computations related to the uniform sampling of LWE and RLWE ciphertexts to be offloaded to the offline phase, a strategy also observed in earlier works such as Simple/Double PIR [32] and FrodoPIR [19]. We notice that we can also employ this technique to achieve a stateless PIR with enhanced throughput. For instance, considering a database configuration of $2^{17} \times 8$ KB (1 GB), the online query and response sizes are 932 KB and 26 KB, respectively, yielding a throughput of 2103 MB/s. Consequently, we can achieve a stateless protocol with high throughput and high rate. More details are presented in Sections 5.3 and 6.5.

2 PRELIMINARY

Notations. We use $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ to denote the set of natural numbers, integers, rational numbers and real numbers, respectively. log refers to the base-2 logarithm. For a positive $k \in \mathbb{Z}$, let $[k]$ be the set of integers $\{0, ..., k-1\}$ and $[a, b]$ be the set $[a, b] \cap \mathbb{Z}$ for any integers $a \le b$. For $x \in \mathbb{R}$, $\lfloor x \rceil$ and $\lfloor x \rceil$ denote the rounding to the lower and closest integer, respectively.

In this paper, a vector is always a column vector by default and is denoted by a bold lower-case letter, e.g., $\boldsymbol{x}$. We use $\boldsymbol{x}[i]$ to denote the i-th element of $\boldsymbol{x}$. For a vector $\boldsymbol{x}$ with k entries, we start the index from 0, i.e., $\boldsymbol{x}[0]$, and the last element is $\boldsymbol{x}[k-1]$. For convenience, we let $\boldsymbol{x}[k] = \boldsymbol{x}[0]$. For a two-dimensional matrix $(m \times n)$ $\mathbf{X}$, $\mathbf{X}[i, j]$ indexes the i-th row and j-th column, where both i, j start from 0. Similar to the one-dimensional case, we set $\mathbf{X}[m, \cdot] = \mathbf{X}[0, \cdot]$ and $\mathbf{X}[\cdot, n] = \mathbf{X}[\cdot, 0]$ for convenience.

We use $\|\boldsymbol{x}\|_\infty$ denotes the l_∞-norm of $\boldsymbol{x}$, i.e., $\|\boldsymbol{x}\|_\infty = \max_i \{\|\boldsymbol{x}[i]\|\}$. For a matrix $\mathbf{X}$, $\boldsymbol{x}_i$ denotes its i-th column vector without extra instructions, $\mathbf{X}^\top$ denotes the transpose of $\mathbf{X}$, $\|\mathbf{X}\|_\infty := \max_i \{\|\boldsymbol{x}_i\|_\infty\}$. Given some set S, $S^{m \times n}$ denotes the set of all $m \times n$ matrices with entries in S.

For a set A and a probability distribution $\mathcal{P}$, we use $a \leftarrow A$ to denote that a is uniformly chosen from A and $a \leftarrow \mathcal{P}$ to denote that a is chosen according to the distribution $\mathcal{P}$.

2.1 Lattice-based Encryptions

Regev introduced the Learning with Errors (LWE) problem [59], whose hardness can be based on some lattice problems. Consider the distribution $A_{\boldsymbol{s}, \chi}$, where χ is a distribution over $\mathbb{Z}$ and $\boldsymbol{s} \in \mathbb{Z}_q^n$ for modulus $q \in \mathbb{N}$. A sample from the distribution $A_{\boldsymbol{s}, \chi}$ is of the form $(b, \boldsymbol{a}) \in \mathbb{Z}_q \times \mathbb{Z}_q^n$, where $\boldsymbol{a} \leftarrow \mathbb{Z}_q^n$, $e \leftarrow \chi$ and $b = \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e \mod q$.

Definition 2.1 (LWE). Let χ be a distribution over $\mathbb{Z}$, $q \geq 2$ be an integer modulus. The decision version of LWE is given m samples with the form of $(b, \boldsymbol{a}) \in \mathbb{Z}_q \times \mathbb{Z}_q^n$ and decide whether these pairs are from the uniform distribution or $A_{\boldsymbol{s},\chi}$.

Another important variant is LWE in the ring setting, known as the Ring Learning with Errors (RLWE) problem [44]. In this work, we only focus on the polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, where n is a power of two, also known as the $2n$-th cyclotomic ring.

For the RLWE problem, now we present the distribution $A_{s,\chi}$. Let χ be a distribution over $\mathcal{R}$ and $s \in \mathcal{R}_q = \mathcal{R}/q\mathcal{R}$, and a sample from $A_{s,\chi}$ is of the form $(b, a) \in \mathcal{R}_q \times \mathcal{R}_q$, where $a \leftarrow \mathcal{R}_q$, $e \leftarrow \chi$ and $b = s \cdot a + e \mod q$. Then, the problem RLWE is presented as:

Definition 2.2 (RLWE). For security parameter λ, let $q = q(\lambda) \geq 2$ be an integer modulus and $\chi = \chi(\lambda)$ be a distribution over $\mathcal{R}$. The task of decision RLWE is, given m pairs of $(b, a) \in \mathcal{R}_q \times \mathcal{R}_q$, decide whether these pairs are from the uniform distribution or $A_{s,\chi}$.

The hardness of the above two problems have been extensively studied in the NIST's post-quantum standardization process in recent years. There are a number of plausible encryption schemes based on LWE or RLWE [8, 9].

In the following, we use four basic encryption schemes based on the LWE or RLWE– (1) LWE, (2) RLWE, (3) RGSW, and (4) RGSW′. Clearly, the security of these schemes can be based on the hard problems of (Ring) Learning with Errors. These schemes have been widely used in the lattice-based cryptography [8, 9] and FHE [10, 15, 21, 23], which imply various homomorphic operations. In the full version of our paper, we present more details about them.

2.2 Polynomial Rings

Here we present some useful notations and properties for the polynomial rings. Let $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where n is a power of two.

Definition 2.3. *Given a ring element $a \in \mathcal{R}$ expressed as $a = a_0 + a_1 X + \cdots + a_{n-1}X^{n-1}$, define* $\mathsf{coef}(a)$ *as the coefficient vector* $(a_0, a_1, \cdots, a_{n-1})$.

Next we present a simple lemma, saying that inner products of the coefficient vectors can be used to capture the constant term by multiplying two ring elements.

Lemma 2.4. *Let $a(X) = a_0 + a_1 X + \cdots + a_{n-1}X^{n-1} \in \mathcal{R}$, $b(X) = b_0 + b_1 X + \cdots + b_{n-1}X^{n-1} \in \mathcal{R}$, and $c(X) = a(X) \cdot b(X)$ with the constant term c_0. Then $c_0 = \langle \mathsf{coef}(a(X)), \mathsf{coef}(b(X^{-1})) \rangle$.*

2.3 Useful Algorithms

Here we recall several useful algorithms for our design: (1) Key-switching, (2) Extract, and (3) Conversion from LWE(s) to RLWE.

Key-switching. The procedure is denoted as KS (Key-switching) with the following algorithms:

- $\mathsf{KS.KeyGen}(1^\lambda, s, s')$. Given two secrets s and s', the algorithm outputs $\mathsf{KSkey} \leftarrow \mathsf{RGSW'.Enc}(\mathsf{sk}, s)$, where $\mathsf{sk} = (1, -s')^\top$.
- $\mathsf{KS}(\mathsf{KSkey}, (b, a))$. Given a RLWE ciphertext $(b, a) \in \mathsf{RLWE}_s(\mu)$ and key-switching key KSkey as input, the algorithm outputs a RLWE ciphertext $(b', a') \in \mathsf{RLWE}_{s'}(\mu)$ by computing

$$(b', a')^\top = (b, 0)^\top - \mathsf{KSkey} \cdot \boldsymbol{g}^{-1}(a).$$

This idea was proposed by [11] and later used widely in the research of FHE [10, 15, 21, 23].

Extract. Given a RLWE ciphertext $(b, a) \in \mathsf{RLWE}_s(\mu)$, there is a simple way that extracts an LWE ciphertext $(b', \boldsymbol{a'}) \in \mathsf{LWE}_{\boldsymbol{s'}}(\mu_0)$, where $\boldsymbol{s'} = \mathsf{coef}(s(X^{-1}))$, $\boldsymbol{a'} = \mathsf{coef}(a)$, $\mu_0 = \mathsf{coef}(\mu)[0]$. This simply follows from Lemma 2.4. The extraction procedure can be easily generalized to outputting an LWE ciphertext that encrypts $\mathsf{coef}(\mu)[i]$ for any i (i.e., any coefficient of μ).

Conversion from LWE*(s) to* RLWE. According to [13, 50], there is an algorithm that given $r(\leq n)$ LWE ciphertexts, namely $(b_0, \boldsymbol{a}_0) \in \mathsf{LWE}_{\boldsymbol{s}}(\mu_0), \cdots, (b_{r-1}, \boldsymbol{a}_{r-1}) \in \mathsf{LWE}_{\boldsymbol{s}}(\mu_{r-1})$ and some proper key-switching key, outputs a RLWE ciphertext $(b, a) \in \mathsf{RLWE}_s(\mu)$ where $\mu = \mu_0 + \mu_1 X + \cdots + \mu_{r-1}X^{r-1}$. We simply call this algorithm r-LWE-to-RLWE.

When $r = O(\log n)$, the method of [13] is faster, whereas when $r = O(n)$, the two methods [13, 50] are roughly the same complexity.

2.4 Private Information Retrieval

Here we describe the syntax of Private Information Retrieval (PIR), following essentially the presentation of [65]. A PIR with preprocessing is a protocol between two stateful machines, namely the *server* and the *client* with the following structure. Implicitly the security parameter 1^λ is taken in all procedures below.

One-time Setup Phase. This phase is run one-time per database and per client. Particularly, the client receives no input and the server receives a database DB of size $N \in \mathbb{N}$, i.e., number of entries. Next the client sends a single message pk (public key) to the server while storing privately the corresponding secret key sk. Then the server does some pre-computation based on DB and pk, resulting in a pair of hints $(\mathsf{hint}_s, \mathsf{hint}_c)$. The server stores hint_s locally and sends back hint_c to the client, as shown in Figure 2.

Query Phase. In this phase, the client would like to retrieve DB[index] for some private index $\in [N]$. This phase can be divided into two stages as follow:

- **(Offline).** The client generates an offline query $\mathsf{qu}_{\mathsf{off}}$ independent of the index, and sends the query to the server. As this step can be done before knowing the querying index, it can be completed in an offline manner.
- **(Online).** Once given the index index $\in [N]$ and the offline query $\mathsf{qu}_{\mathsf{off}}$, the client computes an online query $\mathsf{qu}_{\mathsf{on}}$ and sends the query to the server. Next, the server computes a response r and sends back to the client. Finally given r and sk, the client can then recover the desired DB[index].

There are several variants of PIR that can be captured by the above framework as we discuss in the following remarks.

Remark 2.5. *In general, the Setup phase is run one time per client per database, i.e.,* $(\mathsf{hint}_s, \mathsf{hint}_c)$ *are generated based on the client's* pk *and database* DB.

For any scheme where hint_c *is not needed, e.g.,* hint_c *is an empty string, then the setup is only needed once per client. In this case, the storage required by the client is independent of the number of the databases in the system.*

Next we define several desirable qualities for the offline queries.

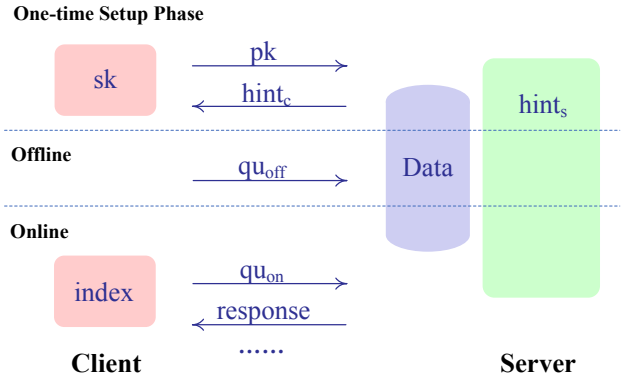

Figure 2: A sketch of Private Information Retrieval.

Definition 2.6. *We say the offline query is* fully-reusable *if one query can be used among multiple online queries over different databases. It is* reusable over multiple databases *if a single* $\mathsf{qu}_{\mathsf{off}}$ *can be used to query multiple databases, but it might not be reused for different online queries.*

In another angle, we define a desirable property – *public-coin.*

Definition 2.7. *The offline query is* public-coin *if the distribution of* $\mathsf{qu}_{\mathsf{off}}$ *follows the uniformly random string that can be sampled publicly (without trapdoor).*

Remark 2.8. *The public-coin property has some desirable practical advantages – in many real-world scenarios, we can just use a random beacon or random oracle to generate public randomness for the query. As long the public source of randomness cannot be controlled by the adversary, this offline query can be made non-interactive where both the client and server just retrieve* $\mathsf{qu}_{\mathsf{off}}$ *by looking at the agreed location at the source. Thus, the server can pre-compute multiple offline queries non-interactively, significantly accelerating the online running time even for multiple queries.*

Next we define correctness and privacy for the PIR scheme.

Correctness. For a database $\mathsf{DB} \in \mathbb{Z}_p^N$, where each element is in $\mathbb{Z}_p$ and can be indexed by a number in $[N]$, the correct answer for a query index $\in [N]$ is the index-th element of DB, denoted as DB[index].

In section 3 and 4, we represent DB with a two dimensional array where index is encoded by two numbers (u, w). In this case, DB[index] refers to DB$[u, w]$ where u is the row index and w is the column index. In section 6, we extend it to handle larger database.

Privacy. We define that a single-server PIR scheme satisfies privacy if and only if there is a probabilistic polynomial-time (PPT) simulator Sim such that for any PPT adversary $\mathcal{A}$ (as the server), any polynomial bound N and Q and any $\mathsf{DB} \in \mathbb{Z}_p^N$, the adversary's view is computationally indistinguishable for the following experiments.

- **Real:** an honest client interacts with $\mathcal{A}(1^\lambda, N, \mathsf{DB})$ who acts as the server but may not follow the prescribed protocol. In the online query stage, for any step $i \in [Q]$, $\mathcal{A}$ may adaptively choose the query $x_i \in [N]$ for the client and the client queries with x_i.
- **Ideal:** the simulated client $\mathsf{Sim}(1^\lambda, N)$ interacts with $\mathcal{A}(1^\lambda, N, \mathsf{DB})$ who acts as the server as the real experiement. In every online step, $\mathcal{A}$ may adaptively choose the query $x_i \in [N]$, and Sim is invoked to generate a simulated query without receiving x_i.

2.5 Faster Matrix-vector Multiplication

Given an $n/2 \times n/2 \times 2$ plaintext matrix $\mathbf{M}$ and an encrypted $n/2 \times 2$-dimensional vector $\boldsymbol{v}$, a commonly used method to homomorphically evaluate matrix vector multiplication is the baby-step giant-step (BSGS) algorithm [30, 34]. The vector $\boldsymbol{v}$ is encoded to plaintext slots and then encrypted to a ciphertext ct. The plaintext matrix is arranged as $n/2$ vectors $\mathbf{M}_{diag}$ that are diagonal rows of $\mathbf{M}$. One BSGS algorithm mainly consists of total $n/2$ plaintext-ciphertext multiplications and $n_1 + n_2$ homomorphic rotations, where $n/2 = n_1 n_2$, as Algorithm 2.1.

By looking deep inside baby-step, we exploit a faster implementation method called *hoisting* [30] without compromising correctness or error growth. $\mathsf{Rotate}(\cdot, i)$ consists of an automorphism transformation $(b_i, a_i) := \left(b(X^{5^i}), a(X^{5^i})\right)$ and a key-switching algorithm $(b_i, 0)^\top - \mathsf{KSkey} \cdot \boldsymbol{g}^{-1}(a_i)$. The automorphism transformation is cheap while key-switching algorithm is much slower as there are some conversions between coefficient and NTT forms.

Algorithm 2.1: BSGS Algorithm for Matrix Vector Multiplication [30, 34]

Input: Inputs an $n/2 \times n/2 \times 2$ plaintext matrix $\mathbf{M}$; a ciphertext ct encrypting vector $\boldsymbol{v}$; two integers n_1 and n_2 such that $n/2 = n_1 n_2$. $\mathbf{M}_{diag}$ are the diagonal rows of $\mathbf{M}$, $\mathbf{M}_{diag}[i]$ is a vector $\left[\mathbf{M}[n/2 - i, 0], \mathbf{M}[n/2 - i + 1, 1], \cdots, \mathbf{M}[n/2 - i - 1, n/2 - 1]\right]$.

Output: Outputs the evaluation result $\bar{ct} = \mathbf{M} \times ct$.

```
// Add, Mul, Rotate are homomorphic addition, multiplication,
   rotation over slots, respectively, ModSwitch is modulus-switching
   algorithm.
for i = 0; i < n1; i++; do
    ct_i = Rotate(ct, i)  // baby step. the ciphertext ct_i encrypting left
       rotated vector of v by i
    ct_i = ModSwitch(ct_i)  // this step is optional
ct̄ := (0, 0)
for j = 0; j < n2; j++ do
    ct_temp := (0, 0)
    for k = 0; k < n1; k++ do
        r = Mul(ct_k, Rotate(M_diag[j · n1 + k], k))  // here Rotate
           acts on plaintext slots
        ct_temp = Add(ct_temp, r)
    ct̄ = Add(ct̄, Rotate(ct_temp, j · n1)) // gaint step
return ct̄
// plaintext can only be encoded to (n/2) × 2-dimensional vector and
   then rotate [14], instead of n-dimensional vector, so the input of this
   algorithm is an n/2 × n/2 × 2 plaintext matrix instead of n × n
   plaintext matrix.
```

In fact, an automorphism transformation on its coefficient form is equivalent to a permutation on its corresponding NTT form [27]. Thus, two NTT representations $\tilde{b}_i = \mathsf{NTT}(b_i)$ and $\tilde{b}_j = \mathsf{NTT}(b_j)$ where $i, j \in [n_1]$, have the same elements, and they are just a

permutation of each other. Therefore, we only have to perform one NTT to obtain $\mathsf{NTT}(b_i)$ for all $i \in [n_1]$, instead of performing n_1 NTTs.

Further, $\boldsymbol{g}^{-1}$ is instantiated as *signed base decomposition* in this work. For a polynomial $a \in \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$, the algorithm $\boldsymbol{g}^{-1} : \mathbb{Z}_q \to \mathbb{Z}^\ell$ outputs a vector $\boldsymbol{x}$ such that $\langle \boldsymbol{g}, \boldsymbol{x} \rangle = a \mod q$, and $\|\boldsymbol{x}\|_\infty \leq \mathcal{B}_\ell/2$, where $\ell := \lceil \log_{\mathcal{B}_\ell} q \rceil$ and base $\mathcal{B}_\ell \in \mathbb{N}$. A useful fact is the property $\boldsymbol{g}^{-1}(-a_j) = -\boldsymbol{g}^{-1}(a_j)$ when using the signed base decomposition algorithm. In the following we notice that $\mathsf{NTT}(\boldsymbol{g}^{-1}(a(X^{5^i}))$ can also be accelerated.

Lemma 2.9. *For a polynomial $a \in \mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ where n is a power of two. $\boldsymbol{g}^{-1}$ is signed base decomposition. we have $\boldsymbol{g}^{-1}(a(X^{5^i})) = \boldsymbol{g}^{-1}(a)(X^{5^i})$.*

Lemma 2.9 states that the automorphism transformations and signed base decomposition are commutative in power-of-two cyclotomic rings, with the proof provided in the full version of our paper. Therefore, we can perform $\boldsymbol{g}^{-1}(a)$ firstly and then perform a series of automorphism transformations. Further, it is known that automorphism transformations can be delayed and replaced by some permutations over NTT representations. So we just do $\mathsf{NTT}(\boldsymbol{g}^{-1}(a))$ and then perform a series of permutations on it, rather than directly computing $\mathsf{NTT}(\boldsymbol{g}^{-1}(a(X^{5^i}))$ for all $i \in [n_1]$. Consequently, n_1 rotations in baby-step (lines 2 and 3 in Algorithm 2.1) can be reduced to one rotation in theory, plus some lightweight permutation operations.

3 OUR FIRST PIR CONSTRUCTION

In the next two sections, we present our new constructions of PIR with pre-processing. We use several recent tools from fully homomorphic encryption, e.g., external products, key-switching, and n-LWE-to-RLWE conversion as Section 2.3. These techniques have been implemented efficiently, and by using them we can achieve more practical protocols.

Building Blocks. We use the following building blocks: (1) RLWE encryption scheme; (2) RGSW and RGSW′ encryption schemes; (3) LWE encryption scheme.

Parameters. Next we describe a list of parameters and symbols used in our PIR constructions.

- $\mathcal{R}$: the underlying ring of the RLWE and RGSW/RGSW′ schemes. In this paper, we set $\mathcal{R} = \mathbb{Z}[X]/(X^n+1)$, where n is a power of two.
- $\mathsf{coef}(a)$: returns the coefficient vector of a, as defined in Lemma 2.3.
- $\mathsf{LWE}_{\boldsymbol{s}}^q$: the set of all legal LWE ciphertexts with the secret $\boldsymbol{s}$ and modulus q(may omitted).
- $\boldsymbol{s}$: the secret of the LWE ciphertexts.
- n, q, p: the dimension of the LWE scheme and the dimension of the ring $\mathcal{R}$, the modulus of the ciphertexts, the modulus of the plaintext space, respectively.
- $\Delta := \lfloor \frac{q}{p} \rfloor$, is the encoding factor.
- $\boxtimes$: the homomorphic external product between a RGSW ciphertext and a RLWE ciphertext, or between a RGSW′ ciphertext and a plaintext.
- $\mathsf{Ext}(b, a)$: returns an LWE ciphertext encrypting the constant term of $m(X)$ under the secret key vector $\mathsf{coef}(s(X^{-1}))$.
- There is a bijective map π between $[N]$ and $[n] \times [n]$. It maps $I \in [N]$ to $(\lfloor I/n \rfloor, I \mod n)$ and the its inverse maps (u, w) to $u \cdot n + w$.
- $\mathsf{Err}(\cdot)$: For any LWE/RLWE ciphertext $\boldsymbol{c}$ or RGSW/RGSW′ ciphertext $\mathbf{C}$, we denote its error by $\mathsf{Err}(\boldsymbol{c})$ or $\mathsf{Err}(\mathbf{C})$, respectively.
- ℓ and $\mathcal{B}_\ell$: are the decomposition dimension and decomposition base in the server's offline phase, respectively.
- ℓ_{exp} and $\mathcal{B}_{exp}$: are the decomposition dimension and decomposition base in the server's online phase, respectively.

3.1 The Construction

Let DB be a database of $N = n^2$ entries represented as a two-dimensional matrix, i.e., $\mathsf{DB} = \{\mathsf{DB}[i,j]\}_{i,j\in[n]}$, where each entry $\mathsf{DB}[i,j] \in \mathbb{Z}_p$. We encode the i-th column of DB using polynomial

$$t_i(X) = \mathsf{DB}[0,i] + \mathsf{DB}[1,i]X + \cdots + \mathsf{DB}[n-1,i]X^{n-1}.$$

Then we present our PIR protocol as follow.

One-time Setup Phase. Below we describe the procedures for the client and the server, respectively.

The **client** receives the security parameter 1^λ and computes:

- Run $(1, -s)^\top \leftarrow \mathsf{RLWE.KeyGen}(1^\lambda)$, and set $\mathsf{sk} = (1, -s)^\top$.
- Store the secret key sk and set $\boldsymbol{s} = \mathsf{coef}(s(X^{-1}))$. We notice that sk can be used as a secret key of RGSW′ as well.
- Then for each $i \in [n]$, generate $\mathsf{pk}_i \leftarrow \mathsf{RGSW'.Enc}(\mathsf{sk}, \boldsymbol{s}[i])$.
- Set $\mathsf{pk} = \{\mathsf{pk}_i\}_{i\in[n]}$ and send pk to the server.

The **server** receives the security parameter 1^λ, database DB, and pk from the client. It stores pk and sets $(\mathsf{hint}_s, \mathsf{hint}_c)$ to be the empty string.

Note. As both the client and server's procedures do not depend on the database, the phase is *one-time per client.*

Query Phase. We now present the offline/online query protocol.

(Offline).

- The **client** samples a uniformly random ring element $a \leftarrow \mathcal{R}_q$, and sends $\mathsf{qu}_{\mathsf{off}} = a$ as the offline query to the server.
- Upon receiving $\mathsf{qu}_{\mathsf{off}} = a$, the **server** does the following:
  (1) For $i \in [n]$, compute $a_i = a \cdot t_i \in \mathcal{R}_q$, and denote $\boldsymbol{a}_i = \mathsf{coef}(a_i)$.
  (2) For $i \in [n]$, set $v_i = v_i(X) = \boldsymbol{a}_0[i] + \boldsymbol{a}_1[i]X + \cdots + \boldsymbol{a}_{n-1}[i]X^{n-1}$.
  (3) Compute $\mathsf{hint}_a = \sum_{i\in[n]} \mathsf{pk}_i \boxtimes v_i$, and store it as the pre-processed information with respect to the query $\mathsf{qu}_{\mathsf{off}} = a$.

Note that hint_a is a RLWE ciphertext that encrypts $m(X) = m_0 + m_1X + \cdots + m_{n-1}X^{n-1}$, where for $i \in [n]$ we have $m_i = \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle$. Recall that $\boldsymbol{s} = \mathsf{coef}(s(X^{-1}))$.

Note. As the offline query is just a uniformly random ring element a, our scheme is public-coin and enjoys the advantages as stated in Remark 2.8.

(Online).

- The **client** receives $\mathsf{index} \in [N]$ and an offline query $\mathsf{qu}_{\mathsf{off}} = a$. It does the following.
  (1) Parse index into $\pi(\mathsf{index}) = (u, w) \in [n] \times [n]$.

(2) Compute $b = s \cdot a + e + \Delta \cdot X^{-u}$. We note that $(b, a) \in \mathsf{RLWE}_s(\Delta \cdot X^{-u})$, where a is the random ring element generated as the offline query.
(3) Compute $\mathbf{C} \leftarrow \mathsf{RGSW.Enc}(\mathsf{sk}, X^{-w})$.
(4) Set $\mathsf{qu}_{\mathsf{on}} = (b, \mathbf{C})$ and send the pair to the server.

- Upon receiving $\mathsf{qu}_{\mathsf{on}} = (b, \mathbf{C})$, the **server** does the following:

(1) For $i \in [n]$, compute $b \cdot t_i \in \mathcal{R}_q$ and set $b_i \in \mathbb{Z}_q$ as its constant term.
(2) Let hint_a be the hint with respect to the offline query a, and compute $\mathsf{ans}_0 = (b_0 + b_1 X + \cdots + b_{n-1} X^{n-1}, 0) - \mathsf{hint}_a$.
(3) Compute $(b', a') = \mathbf{C} \boxtimes \mathsf{ans}_0$.
(4) Send response $\mathsf{r} = (\overline{b}, \overline{\boldsymbol{a}}) = \mathsf{Ext}(b', a')$ back to the client.

- Upon receiving the response r, the **client** outputs

$$d = \mathsf{LWE.Dec}(\mathsf{coef}(s(X^{-1})), (\overline{b}, \overline{\boldsymbol{a}})).$$

Additional analysis. We present the error growth, correctness and security analysis in the full version of our paper.

4 OUR SECOND PIR CONSTRUCTION

In this section, we present our second construction, by leveraging additional pre-processing to improve the server's offline query response. Particularly, the server is doing one-time setup per database per client and store some hint_s, which can be used to accelerate the response time for each offline query. The client is still doing one-time setup per client.

One-time Setup Phase. Below we describe the procedures. Here we note that for better noise control, we use two ciphertext modulus $Q > q$ and the modulus switching technique [10]. For conceptual understanding of the protocol, the reader can just think of $Q = q$.

The **client** receives the security parameter 1^λ and does exactly the same as the prior construction but with a larger modulus Q. We re-state the process.

- Run $(1, -s)^\top \leftarrow \mathsf{RLWE.KeyGen}(1^\lambda)$, and set $\mathsf{sk} = (1, -s)^\top$.
- Store the secret key sk and set $\boldsymbol{s} = \mathsf{coef}(s)$. We notice that sk can be used as the secret key of RGSW′ as well.
- Then for each $i \in [n]$, generate $\mathsf{pk}_i \leftarrow \mathsf{RGSW'.Enc}(\mathsf{sk}, \boldsymbol{s}[i])$.
- Set $\mathsf{pk} = \{\mathsf{pk}_i\}_{i \in [n]}$ and send pk to the server.

The **server** receives the security parameter 1^λ, database DB, and pk from the client. Then it does the following:

- Let $v_j = \sum_{k \in [n]} \mathsf{DB}[j, k] \cdot X^k$. For $i \in [n]$, compute $\mathsf{KSkey}'_i = \sum_{i'=0}^{i} \mathsf{pk}_{i'} \cdot v_{i-i'} - \sum_{i'=i+1}^{n-1} \mathsf{pk}_{i'} \cdot v_{n+i-i'}$
- $\mathsf{KSkey}_i = \lfloor \mathsf{KSkey}'_i \cdot \frac{q}{Q} \rceil \mod q$. The server sets hint_c as empty string and $\mathsf{hint}_s = \{\mathsf{KSkey}_i\}_{i \in [n]}$.

We note that each KSkey_i is a RGSW′ ciphertext that encrypts $\sum_k X^k \cdot \boldsymbol{s}_k[i]$, if each vector $\mathsf{coef}(s \cdot t_k)$ is denoted as $\boldsymbol{s}_k$ for $k \in [n]$.

Note. Clearly, for the client this process is one-time per client; for the server, this is one-time per client per database.

Query Phase. We now present our protocol for the offline/online query phase.

(Offline).

- The **client** samples a uniformly random ring element $a \leftarrow \mathcal{R}_q$, and sends $\mathsf{qu}_{\mathsf{off}} = a$ as the offline query to the server.
- Upon receiving pk and $\mathsf{qu}_{\mathsf{off}} = a$, the **server** computes and stores the pre-processed information with respect to the query $\mathsf{qu}_{\mathsf{off}} = a$ as

$$\mathsf{hint}_a = \sum_{i \in [n]} \mathsf{KSkey}_i \boxtimes \boldsymbol{a}[i],$$

where we denote $\mathsf{coef}(a(X^{-1}))$ as $\boldsymbol{a}$.

Note. The same as our first construction, our second scheme is public-coin and enjoys the advantages as stated in Remark 2.8.

(Online). The online phase is exactly the same as the prior construction in Section 3, so we omit the detailed description here.

Additional analysis. Similar to Section 3, we present the error growth, correctness and security analysis in the full version of our paper.

5 OUR THIRD PIR CONSTRUCTION

In this section, we present another construction that delivers improved overall runtime (offline + online) without requiring preprocessing compared to the prior two schemes. As an interesting tradeoff, the online time of this scheme is slightly longer than that in the previous schemes.

We notice that both our first two constructions contain the first dimension folding operation to extract a certain column (or row) from the given database matrix. From the linear algebra expression, these schemes actually (homomorphically) compute

$$\mathsf{DB} \cdot \boldsymbol{u},$$

where $\boldsymbol{u}$ is some vector with only one non-zero entry 1. As pointed in the BGV/BFV/CKKS bootstrapping and ciphertext transformations [34], we can efficiently compute the product of a matrix and a vector by the BSGS algorithm (ref. Section 2.5). We treat the BSGS as a blackbox algorithm, i.e., $\mathsf{BSGS}(\mathbf{M}, \boldsymbol{c})$ takes a plaintext matrix $\mathbf{M}$ and a cihpertext $\boldsymbol{c} \in \mathsf{RLWE}(\Delta \cdot \mathsf{NTT}^{-1}(\boldsymbol{u}))$ as input and outputs a $\mathsf{RLWE}(\Delta \cdot \mathsf{NTT}^{-1}(\mathbf{M} \cdot \boldsymbol{u}))$ ciphertext. Using this as a building block, we present our third scheme as follow.

5.1 The Construction

Let DB be a database of $N = n/2 \times n/2 \times 2$ entries represented as a two-dimensional matrix, i.e., $\mathsf{DB} = \{\mathsf{DB}[i, j]\}_{i \in [n/2], j \in [n/2]}$, where each entry $\mathsf{DB}[i, j] \in \mathbb{Z}_p^2$ (plaintext can only be encoded to $(n/2) \times 2$-dimensional vector and then rotate [14], so each slot have two entries $\in \mathbb{Z}_p$). Apply NTT to each column of DB to get $\mathbf{M}$. Then our PIR works as follow.

One-time Setup Phase. Below we describe the procedures for the client and the server, respectively.
The **client** receives the security parameter 1^λ and does the following:

- Run $(1, -s)^\top \leftarrow \mathsf{RLWE.KeyGen}(1^\lambda)$, and set $\mathsf{sk} = (1, -s)^\top$.
- Store key-switching keys used in the homomorphic automorphisms in the BSGS procedure in pk.

The **server** receives the security parameter 1^λ, database DB, and pk from the client. The server does nothing.

Query Phase. We now present the query protocol.

- The **client** receives index $\in [N]$ and does the following.
  (1) Parse index into $\pi(\text{index}) = (u, w) \in [n/2] \times [n]$. Denote the vector with only one non-zero entry 1 in the u-th position as $\boldsymbol{u}$.
  (2) Randomly choose a from R_q. Compute $b = s \cdot a + e + \Delta \cdot \mathsf{NTT}^{-1}(\boldsymbol{u})$. We note that $(b, a) \in \mathsf{RLWE}_s(\Delta \cdot \mathsf{NTT}^{-1}(\boldsymbol{u}))$[1].
  (3) Compute $\mathbf{C} \leftarrow \mathsf{RGSW.Enc}(\mathsf{sk}, X^{-w})$.
  (4) Set $\mathsf{qu} = ((b, a), \mathbf{C})$ and send the pair to the server.
- Upon receiving $\mathsf{qu} = ((b, a), \mathbf{C})$, the **server** does the following:
  (1) compute $\mathsf{ans}_0 = \mathsf{BSGS}(\mathbf{M}, (b, a))$.
  (2) Compute $(b', a') = \mathbf{C} \boxtimes \mathsf{ans}_0$.
  (3) Send response $\mathsf{r} = (\overline{b}, \overline{\boldsymbol{a}}) = \mathsf{Ext}(b', a')$ back to the client.
- Upon receiving the response r, the **client** outputs

$$d = \mathsf{LWE.Dec}(\mathsf{coef}(s(X^{-1})), (\overline{b}, \overline{\boldsymbol{a}})).$$

5.2 Analysis

We present Theorem 5.1 to illustrate the correctness of our third construction.

Theorem 5.1 (Correctness). *Adopt the notations from the above section. If the parameter q keeps the* LWE.Dec *correct, then for any input query (u, w), the final output of our protocol satisfies $d = \mathsf{DB}[w, u]$.*

Proof. We begin by analyzing the underlying plaintexts at each step of the homomorphic computation in our construction. Then in the full version of our paper, we estimate noise growth and calculate the probability of decryption failure, thereby validating the correctness of our overall scheme.

In the query phase, we mainly focus on the server. First note that (b, a) is a RLWE ciphertext encrypting $\Delta \cdot \mathsf{NTT}^{-1}(\boldsymbol{u})$ with secret s. By the property of BSGS, we have ans is a RLWE encryption of $\mathsf{NTT}^{-1}(\mathbf{M} \cdot \boldsymbol{u})$. In the pre-processing, we set $\mathbf{M} = \mathsf{NTT}(\mathsf{DB})$ in advance. Moreover, $\boldsymbol{u}$ is a vector with its only non-zero element being 1 in the u-th position, and multiplying it from the right is equivalent to extracting the u-th column. Therefore, we have

$$\mathbf{M} \cdot \boldsymbol{u} = \mathsf{NTT}(\mathsf{DB}) \cdot \boldsymbol{u} = \mathsf{NTT}(\mathsf{DB} \cdot \boldsymbol{u}).$$

In step (2), the server performs an external product, so (b', a') is an RLWE ciphertext that encrypts $X^{-w} \cdot (\sum_i \Delta \cdot \mathsf{DB}[i, u] X^i)$. In step (3), the server applys $\mathsf{Ext}(\cdot)$, so the result r is an LWE ciphertext that encrypts $\Delta \cdot \mathsf{DB}[w, u]$.

Through the above analysis, we have $d = \mathsf{DB}[w, u]$ by the correctness of LWE.Dec. □

5.3 Stateless Variant

Motivated by recent advancements in PIR protocols including HintlessPIR [41], YPIR [49] and WhisPIR [20], we derive a *stateless* variant of our third construction. In this setting, not only are the clients free from storing any database-dependent hint, but the server also avoids retaining any client-specific storage, e.g., key-switching keys. This is achieved by having clients upload key-switching key materials during the online phase, instead of the server storing them for each client. While this approach increases the query size, it significantly reduces server-side storage requirements, simplifies deployment, and enhances anonymity by obscuring which client is querying at any given time [41].

Moreover, in most FHE-based protocols, the query ciphertexts must be homomorphically multiplied with the database. Whether the query ciphertext is the LWE ciphertext $(b, \boldsymbol{a})$ used in Simple/Double PIR [32] and FrodoPIR [19], or the RLWE ciphertext (b, a) used in Spiral, using the same $\boldsymbol{a}$ (resp. a) to encrypt multiple messages —while independently sampling $\boldsymbol{s}$ (resp. s) and error e —does not compromise security [57]. In Simple/Double PIR and FrodoPIR, the component $\boldsymbol{a}$ of the query ciphertext $(b, \boldsymbol{a})$ is preprocessed to enhance efficiency during the online phase. Subsequent works HintlessPIR [41], YPIR [49] and WhisPIR [20] extend this insight to key-switching keys, i.e., the components $\boldsymbol{a}^\top$ in key-switching keys $(\boldsymbol{b}^\top; \boldsymbol{a}^\top)$ can also be preprocessed. Following this insight, we can achieve the stateless variant and below we describe how it works.

One-time Setup Phase. Below we describe the procedures for the client and the server, respectively.
The **server** receives the security parameter 1^λ, database DB, and does the following:

- Sample a PRG seed $\sigma \leftarrow \{0, 1\}^\lambda$.
- Expand the seed σ to generate dummy RLWE ciphertext $(0, a)$, and dummy key-switching keys in $\overline{\mathsf{pk}} = \{\overline{\mathsf{KSkey}_i} = (\mathbf{0}^\top; \boldsymbol{a}_i^\top) \in \mathcal{R}_q^{2\times\ell}, i \in [1 + (n_2 - 1)]\}$, where $\{\overline{\mathsf{KSkey}_i}\}$ are the required keys in the BSGS procedure.
- Compute $\overline{\mathsf{ans}_0} = \mathsf{BSGS\_setup}(\mathbf{M}, (0, a))$ [2],

The **client** receives the security parameter 1^λ and does nothing.

Query Phase. We now present the query protocol.

- The **client** receives the security parameter 1^λ, the public seed σ and does the following:
  (1) Run $(1, -s)^\top \leftarrow \mathsf{RLWE.KeyGen}(1^\lambda)$, and set $\mathsf{sk} = (1, -s)^\top$.
  (2) Expand the seed σ to generate key-switching keys $\{\mathsf{KSkey}_i = (\boldsymbol{b}_i^\top; \boldsymbol{a}_i^\top) \in \mathcal{R}_q^{2\times\ell}, i \in [n_2]\}$ used in the BSGS procedure, and store $\widetilde{\mathsf{pk}} = \{\widetilde{\mathsf{KSkey}_i} = (\boldsymbol{b}_i^\top; \mathbf{0}^\top), i \in [n_2]\}$.
- Upon receiving index $\in [N]$, the **client** does the following:
  (1) Parse index into $\pi(\text{index}) = (u, w) \in [n/2] \times [n]$. Denote the vector with only one non-zero entry 1 in the u-th position as $\boldsymbol{u}$.
  (2) Expand the seed σ to ring element a in $\mathcal{R}_q$ and compute $b = s \cdot a + e + \Delta \cdot \mathsf{NTT}^{-1}(\boldsymbol{u})$. We note that $(b, a) \in \mathsf{RLWE}_s(\Delta \cdot \mathsf{NTT}^{-1}(\boldsymbol{u}))$.
  (3) Compute $\mathbf{C} \leftarrow \mathsf{RGSW.Enc}(\mathsf{sk}, X^{-w})$.
  (4) Set $\mathsf{qu} = ((b, 0), \mathbf{C}, \widetilde{\mathsf{pk}})$ and send it to the server.

[1] In our implementation we use $\mathsf{RLWE}_s(\lfloor q/p \cdot \mathsf{NTT}^{-1}(\boldsymbol{u}) \rceil)$ thus to achieve smaller error growth [35].

[2] For a smaller key-switching key size, we do not employ the hoisting technique described in Section 2.5. Instead, we use the iterative rotation method detailed in HintlessPIR [41] and Figure 8, Appendix A.3 of WhisPIR [20].

- Upon receiving $\mathsf{qu} = ((b,0), \mathbf{C}, \widetilde{\mathsf{pk}})$, the **server** does the following:
  (1) Compute $\mathsf{ans}_0 = \overline{\mathsf{ans}}_0 + \mathsf{BSGS\_online}(\mathbf{M}, (b,0))$.
  (2) Compute $(b', a') = \mathbf{C} \boxtimes \mathsf{ans}_0$.
  (3) Send response $\mathsf{r} = (\overline{b}, \overline{\boldsymbol{a}}) = \mathsf{Ext}(b', a')$ back to the client.
- Upon receiving the response r, the **client** outputs

$$d = \mathsf{LWE.Dec}(\mathsf{coef}(s(X^{-1})), (\overline{b}, \overline{\boldsymbol{a}})).$$

The stateless variant differs from the original protocol in two significant ways: 1). The client uploads key-switching keys used in the homomorphic automorphisms during BSGS procedure in the online phase, while the server preprocesses all public components across all clients. 2). We employ the iterative rotation method detailed in HintlessPIR [41] and WhisPIR [20] in the baby-step algorithm instead of using the hoisting technique.

Recalling that in the baby-step algorithm, the server rotates the same ciphertext ct by 1 to $n_1 - 1$. Although the hoisting technique reduces the theoretical computational complexity from $n_1 - 1$ to 1, the key-switching keys remain at $n_1 - 1$. The iterative rotation, i.e., $\mathsf{Rotate}^{(n_1-1)}(ct, 1)$, is more suitable in the stateless setting, as it only requires a single key-switching key. Most importantly, HintlessPIR and WhisPIR found that the heaviest computations can be preprocessed. We introduce the principle of BSGS_setup and BSGS_online here, and the detailed iterative rotation algorithm can be referred to Figure 8 of WhisPIR [20]: It is known that $\mathsf{Rotate}(\cdot, 1)$ consists of an automorphism transformation $(b_1, a_1) := \big(b(X^5), a(X^5)\big)$ and a key-switching algorithm $ct_1 = (b_1, 0)^\top - \mathsf{KSkey}_0 \cdot \boldsymbol{g}^{-1}(a_1)$. In fact, most heavy computation can be offloaded to the one-time setup phase. We assume that the dummy key-switching key is $\overline{\mathsf{KSkey}}_0 = (\mathbf{0}^\top; \boldsymbol{a}_0^\top)$ in the one-time setup phase. The server preprocesses $(0, a_1) := \big(0, a(X^5)\big)$ and $\overline{ct}_1 = (0,0)^\top - \overline{\mathsf{KSkey}_0 \cdot \boldsymbol{g}^{-1}(a_1)}$ in the one-time setup phase. Upon receiving $(b, 0)$ and $\widetilde{\mathsf{KSkey}}_0 = (\boldsymbol{b}_0^\top; \mathbf{0}^\top)$ in the online phase, the server computes $(b_1, 0) := \big(b(X^5), 0\big)$ and $ct_1 = \overline{ct}_1 + (b_1, 0)^\top - \widetilde{\mathsf{KSkey}_0 \cdot \boldsymbol{g}^{-1}(a_1)}$. Noting that $\mathsf{KSkey}_0 = \overline{\mathsf{KSkey}}_0 + \widetilde{\mathsf{KSkey}}_0$, the ciphertext ct_1 is precisely the rotated ciphertext by 1. The server also store $\boldsymbol{g}^{-1}(a_1)$ in NTT form, and the automorphism transformation $b(X^5)$ can be directly preformed in its NTT form [27], thus the online phase can be greatly accelerated. By using this method for iterative rotations and extending the precomputation approach to plaintext-ciphertext multiplication and the giant step, the throughput wound be much better than that of the original construction.

6 IMPLEMENTATION AND EVALUATION

We implement our protocols in C++ to evaluate their concrete efficiency. Our implementations do not use any existing FHE library but adopt the Intel HEXL library (v1.2.5) to implement the NTTs. The source code is available at [43].

6.1 Extensions and Optimizations

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

Handling Larger Database. In the last three sections we introduced three constructions, both of which can support the *basic* database of n^2 records ($n^2/2$ records in third construction, below we may ignore that), where the size of each record is $\log p$. Now we describe how to extend them to support larger databases and longer records. 1) larger databases: there are only $n^2 \times \log p$ bits in one basic database, and the value is about 16 MB to 36 MB under concrete parameters. The server can do r times for r basic databases, with the same query. Finally the server can use a r-LWE-to-RLWE algorithm [13] to pack r LWE ciphertexts. 2) longer records: $r \log p$ bits may not be enough for some applications. We use two approaches to accommodate more bits. One is the more general usage of r-LWE-to-RLWE algorithm. The subprocedure (line 5-7 in PackLWEs algorithm [13]) can packing two RLWE ciphertexts $\mathsf{RLWE}(\sum_{i\in[n]} m_i X^i)$ and $\mathsf{RLWE}(\sum_{i\in[n]} \tilde{m}_i X^i)$ to a new ciphertext $\mathsf{RLWE}(\sum_{i\in[n]} \bar{m}_i X^i)$, such that $\bar{m}_{kn/\ell} \approx m_{kn/\ell}$ and $\bar{m}_{kn/\ell+n/(2\ell)} \approx \tilde{m}_{kn/\ell}$ for $k \in [\ell]$ and ℓ is a power of two, with only small noise growth produced by evaluating automorphic transformation. Therefore, the server don't always extract LWE ciphertexts and packing them but just directly packing r RLWE ciphertexts and response. Finally, each record is $n \log p$ bits, which is around 8 KB under concrete parameters. To the best of our knowledge, using external product and subprocedure of r-LWE-to-RLWE algorithm [13] to handle a small number of RLWE ciphertexts is unprecedented in previous PIR protocols. Further, we also use the same approach as SpiralPack [46] when the records are larger than 8 KB, i.e., packing multiple RLWE ciphertexts to a matrix Regev ciphertext, thus to achieve better rate.

Improvement by Approximate Decomposition. We also use another variant of algorithm $\boldsymbol{g}^{-1}$ in our implementation. Given a modulus q and $\ell := \lceil \log_{\mathcal{B}_\ell} q/\mathcal{B}_e \rceil$, we denote the gadget vector as $\boldsymbol{g}^\top = \mathcal{B}_e \cdot (1, \mathcal{B}_\ell, ..., \mathcal{B}_\ell^{\ell-1})$ for some base $\mathcal{B}_\ell, \mathcal{B}_e \in \mathbb{N}$. Then we use the algorithm $\boldsymbol{g}^{-1} : \mathbb{Z}_q \to \mathbb{Z}^\ell$, such that the output of the algorithm $\boldsymbol{x} \leftarrow \boldsymbol{g}^{-1}(a)$ satisfy $\langle \boldsymbol{g}, \boldsymbol{x} \rangle \approx a \mod q$. The approximate gadget decomposition is first used for the torus variant of LWE and RLWE samples in TFHE [15]. It also works well in our protocol. In the following we call ℓ, $\mathcal{B}_\ell$ and $\mathcal{B}_e$ as decomposition dimension, decomposition base and approximate base, respectively.

Reducing Size of Response. Like other lattice-based PIR protocols [2, 46], the server of our protocol performs a modulus switching in order to reduce the size of response. Given a RLWE ciphertext (b, a), the server computes $\lfloor q_{mod} \cdot (b, a)/q \rceil \mod q_{mod}$, where q_{mod} is a smaller modulus than q.

Reducing Size of Public Key. Both in the first and the second constructions, the client sends a public key pk to the server. Note that $\mathsf{pk} = \{\mathsf{pk}_i\}_{i\in[n]}$ and each pk_i is a RGSW′ ciphertext that encrypts only a constant term. We can use an expansion algorithm to reduce communication, like SealPIR [4] and Spiral [46]. The processing is performed once for each client, and it can be reused for arbitrary databases.

Heuristic Noise Analysis with Subgaussian Variables. We estimate the noise growth in the way of independent subgaussian variables. This type of estimate is tighter than the bounding of worst-case noise magnitude, and it's also closer to what we observe in practice. All of our following experiments in this paper have been checked with lower than decryption failure probability 2^{-40}. We put the concrete analysis in the full version of our paper.

Database	Metric	FastPIR	Spiral	SpiralPack	Spiral StreamPack	First construction	Second construction	Third construction
$(2^{20}+2^{17})\times 32$B 36 MB	One-time setup	0	0	0	0	0	879 s	0
	Offline comp.	0	0	0	0	638 ms	93 ms	0
	Online							
	Server response time	538 ms	637 ms	548 ms	165 ms	14.5 ms	14.5 ms	77.1 ms
	Query size	192 KB	14 KB	14 KB	2.4 MB	25 KB	25 KB	140 KB
	Response size	64 KB	20 KB	20 KB	20 KB	28 KB	29 KB	26 KB
	Throughput	67 MB/s	57 MB/s	66 MB/s	218 MB/s	2483 MB/s	2483 MB/s	467 MB/s
$^{\dagger}2^{20}\times 256$B 256 MB	One-time setup	0	0	0	0	0	6972 / ‡672 s	0
	Offline comp.	0	0	0	0	5112 ms	746 ms	0
	Online							
	Server response time	876 ms	1744 ms	1424 ms	524 ms	133.1 ms	133.1 ms	243 ms
	Query size	1 MB	14 KB	14 KB	7.8 MB	125 KB	125 KB	140 KB
	Response size	64 KB	20 KB	20 KB	20 KB	24 KB	27 KB	26 KB
	Throughput	292 MB/s	147 MB/s	180 MB/s	489 MB/s	1923 MB/s	1923 MB/s	1053 MB/s
$2^{22}\times 256$B 1 GB	One-time setup	0	0	0	0	0	‡2953 s	0
	Offline comp.	0	0	0	0	20 s	2955 ms	0
	Online							
	Server response time	2306 ms	3628 ms	3094 ms	1756 ms	694.2 ms	694.2 ms	801 ms
	Query size	4 MB	14 KB	14 KB	15 MB	135 KB	135 KB	140 KB
	Response size	64 KB	20 KB	20 KB	20 KB	28 KB	26 KB	26 KB
	Throughput	444 MB/s	282 MB/s	331 MB/s	583 MB/s	1475 MB/s	1475 MB/s	1278 MB/s

Table 2: Compared with Spiral. † Our database configuration is $2^{15}\times 8$ KB, which can be trivially seen as $2^{20}\times 256$ B. ‡Running in $T=16$ threads, whereas all the others are run in $T=1$ thread. For 256 MB database of Spiral, we use the default parameters in [47]. We adjust the parameters of Spiral to get 36 MB, 1 GB database configuration, e.g., v_1 (the number of the first dimension), v_2 (the number of the second dimension), $\log p$ (plaintext bits), and so on. For 36 MB, we use $(v_1, v_2, \log p) = (7, 5, 9)$. For 256 MB, we use $(v_1, v_2, \log p) = (8, 7, 8)$. For 1 GB, we use $(v_1, v_2, \log p) = (9, 8, 8)$.

6.2 Parameter Selection

In this section, we present how we select parameters for our schemes.

Lattice parameters. Our protocols always work over a power-of-two cyclotomic ring. In order to ensure 128 bits of classical security and take the noise growth into account, we set ring dimension $n = 4096$. Each secret is sampled as ternary secret with the Hamming weight h, and all the initial noise is sampled from discrete Gaussian distribution with standard deviation $\sigma = 3.19$.

Database and Key Material. The database is arranged as a hypercube with dimensions $n^2 \times r$, where each n^2 elements form a basic database, r is the packing number. Our database is stored in its evaluation representation (i.e., the FFT/NTT representation). This enables faster homomorphic operations during online query processing. Similarly, the automorphism transform key materials are all stored in their evaluation representation.

6.3 Concrete Performances for Our PIR Protocols

In this section, we report the concrete performances of our protocols. Our computing environment is a server with Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz and 256GB RAM, running Ubuntu 22.04.1. The compiler we used is clang++ 14.0.0.

Experimental Results. We list four different database configurations, which are 36 MB, 256 MB, 1 GB and 2 GB, and we provide concrete experimental results to confirm efficiency of our protocols in Table 3 and Table 2.

Comparison of Three Constructions. In the previous three sections, we presented three PIR constructions. The first and second constructions have better online response time. They share the same online phase processing, but they have their own advantages and disadvantages in the one-time setup phase and offline phase. The first construction has a longer offline processing time while the second construction has a longer one-time setup phase.

Table 3 shows the concrete experimental results of the third construction for different database configurations. We find that BSGS algorithm consumes a relatively large proportion of time. In fact, due to the use of hoisting technique (see Section 2.5) and the output of the baby-step algorithm can be reused by multiple basic databases, most of the consumption of the BSGS algorithm is plaintext-ciphertext multiplications, i.e., plaintext-ciphertext multiplications seem to be intrinsic for PIR protocols belonging to Category I.

	Metric	36 MB	256 MB	1 GB	2 GB
Communication (KB)	Query size	140	140	140	140
	Response size	26	26	26	26
Client cost (ms)	Query	5.9	6.1	6.2	6
	Recover	1.5	1.5	1.4	1.5
Server cost (ms)	Offline	0	0	0	0
	First dim.	72.9	228.6	742.6	1391
	Second dim.	1.7	6.4	23.6	42.5
	Packing	2.4	8.3	34.4	64
	Online total	77.1	243	801	1498
Throughput (MB/s)		467	1053	1278	1367

Table 3: Concrete experimental results of the third construction for different database configurations. "dim." stands for dimension. BSGS algorithm is called *the first dimension folding*. External product is called *the second dimension folding*. Packing multiple ciphertexts is called *the packing algorithm*.

6.4 Comparisons with other protocols in Category I and Category II

We discuss comparisons of our three constructions with other related work in Category I, e.g., Spiral and its prior works. We also discuss comparisons with other works in Category II, e.g., Simple/Double PIR [32] and Piano [65].

Compared with Spiral and its Prior Works. The single-server PIR scheme Spiral [46] follows the Gentry-Halevi [26] blueprint. They rely on two basic encryption schemes: RLWE encryption scheme and the RGSW encryption scheme. After a query expansion phase, the server performs plaintext-ciphertext multiplications in the first dimension folding. Then the server uses external product to perform ciphertext-ciphertext multiplications, which is also called the subsequent dimensions folding. Thanks to the expansion algorithm and low noise growth of external product, Spiral as well as its family outperform almost all other lattice-based PIR protocols, e.g., SealPIR [4], FastPIR [1], MulPIR [2], OnionPIR [52]. Therefore, Spiral can be considered as the current state of the art, and a baseline for us to compare. For a fair comparison, we use their C++ implementation, which adapts procedure from the SEAL library and HEXL library to implement NTTs. The comparison results are given in the Table 2 and Figure 3. Taking the performance of 256 MB database as an example, our server's online response time of third construction is 5.8× faster than SpiralPack.

Compared with Simple/Double PIR and more. Another well-known type of PIR protocols are proposed by A. Henzinger et al. [32], which are called Simple PIR and Double PIR. As PIR protocols in Category II, the biggest advantage is their high throughput. The drawback of them is that the client must download and store an around 124 MB (Simple PIR) and 16 MB hint (Double PIR), respectively. Meanwhile, larger records (e.g., 256 bits) are not friendly to Double PIR, so there are some limitations in applying it to many application scenarios.

Simple PIR can achieve more than 6000 MB/s server throughput while Double PIR can achieve more than 5000 MB/s. However, our PIR protocol is more client cost friendly, both in terms of computation and storage. The client only needs to store the secret and a public coin. We show the comparison in Figure 3 and defer more in the full version of our paper. Besides, the client in Simple PIR and Double PIR should update hint when the database updates. In our setting, database updating has nothing to do with the state of the client.

Other Related Works. Besides the practical PIR protocols which we have mentioned, e.g., XPIR [45], SealPIR [4], SHECS-PIR [54], FastPIR [1], MulPIR [2], OnionPIR [52] and Spiral [46], multi-server PIR and PIR for special scenarios have also received great attentions. Currently the most practical two-server PIR comes from Corrigan-Gibbs and Kogan et al. [18, 36]. They propose a new two-server PIR [18] that can achieve sublinear online time, and they improve it and publish experimental results [36]. Special PIR includes SparsePIR [56], which is an efficient keyword PIR for sparse databases.

6.5 Comparisons with Concurrent Stateless Protocols

Recently, Li et al. [41] proposed HintlessPIR based on Simple PIR. They found that the clients do not need to store the hint in Simple PIR [32], but online homomorphic evaluate matrix-vector multiplication [29], involving the hint matrix and secret vector. Furthermore, this RLWE-based matrix-vector multiplication can still be accelerated by offloading part of the computation to the offline phase, similar to the precomputation approach in Simple/Double PIR [32] and FrodoPIR [19].

Menon and Wu [49] adopted an approach similar to HintlessPIR. The starting point for their construction is Double PIR. They found that the hint and online response in Double PIR consist of many LWE ciphertexts. Utilizing an LWE-to-RLWE conversion algorithm [13], YPIR enables to response the RLWE ciphertext instead of many coefficients and storing a hint matrix on the client side.

Castro et al. [20] proposed WhisPIR, a stateless protocol characterized by low communication overhead. In WhisPIR, clients upload key-switching keys during the online phase, and the protocol optimizes the key-switching key size in the coefficient expansion algorithm used by SealPIR [4], OnionPIR [52] and Spiral [46]. Following an optimized coefficient expansion algorithm, the server performs plaintext-ciphertext multiplications and then non-compact homomorphic multiplications, i.e., tensor multiplication without relinearization.

These three stateless protocols as well as Spiral (and its related family) are currently state of the art in Category I. In Table 4, we collect concrete experimental results for ours and these three protocols, as well as Simple/Double PIR (in Category II), to show their unique advantages. All protocols (expect for WhisPIR) are tested under the same computing environment specified in Section 6.3, running on a single thread. The total database sizes are 256 MB, 1 GB and 8 GB, with the record size being optimal and most recommended for each protocol.

Compared with HintlessPIR and YPIR. HintlessPIR and YPIR are two stateless LWE-based PIR protocols derived from Simple and Double PIR, respectively. When retrieving large databases, their

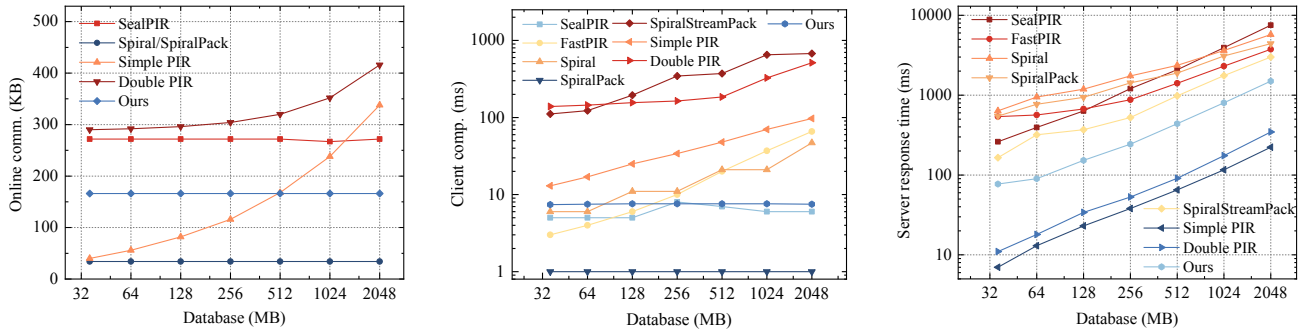

Figure 3: Comparison of online communication, client computation and sever response time of SealPIR, FastPIR, Spiral, SpiralPack, SpiralStreamPack, Simple/Double PIR and Ours third construction. The online communication of FastPIR, SpiralStreamPack is much bigger than others.

Database	Metric	Spiral (8 KB)	Simple PIR† (16 KB)	Double PIR† (1B)	HintlessPIR (1B, 32 KB)	YPIR (1B)	YPIR + SP (32 KB)	WhisPIR (256 B, 32 KB)	Ours (8 KB)	Ours-stateless (8 KB)
256 MB	Server storage	13.3 MB	0	0	0	0	0	NA	8.8 MB	0
	Hint size	0	56 MB	14 MB	0	0	0	NA	0	0
	Server preproc.	9.7 s	6.4 s	9.8 s	53.2 s	9.8 s	13.1 s	NA	4.3 s	6.6 s
	Server resp.	1744 ms	28 ms	36 ms	598 ms	86 ms	314 ms	NA	243 ms	149 ms
	Query size	14 KB	64 KB	112 KB	379 KB	846 KB	518 KB	NA	140 KB	988 KB
	Response size	20 KB	56 KB	12 KB	1504 KB	12 KB	120 KB	NA	26 KB	26 KB
	Throughput	147 MB/s	9143 MB/s	7111 MB/s	428 MB/s	2977 MB/s	815 MB/s	NA	1053 MB/s	1718 MB/s
	Client comp.	11 ms	234 ms	914 ms	1928 ms	977 ms	323 ms	NA	7.6 ms	42.2 ms
1 GB	Server storage	13.6 MB	0	0	0	0	0	0	9.0 MB	0
	Hint size	0	112 MB	14 MB	0	0	0	0	0	0
	Server preproc.	40 s	26 s	31.6 s	187 s	31.6 s	21.5 s	NA	16.9 s	19.2 s
	Server resp.	3628 ms	111 ms	126 ms	866 ms	178 ms	426 ms	≈1000 ms	801 ms	487 ms
	Query size	14 KB	128 KB	224 KB	443 KB	846 KB	686 KB	≈390 KB	140 KB	932 KB
	Response size	20 KB	112 KB	12 KB	3008 KB	12 KB	120 KB	≈100 KB	26 KB	26 KB
	Throughput	282 MB/s	9225 MB/s	8127 MB/s	1182 MB/s	5753 MB/s	2404 MB/s	≈1024 MB/s	1278 MB/s	2103 MB/s
	Client comp.	21 ms	459 ms	1988 ms	3958 ms	2050 ms	1223 ms	NA	7.6 ms	39.2 ms
8 GB	Server storage	15.5 MB	0	0	0	0	0	0	9.3 MB	0
	Hint size	0	224 MB	14 MB	0	0	0	0	0	0
	Server preproc	489 s	221 s	232.8 s	1640 s	232.8 s	93 s	NA	148 s	157 s
	Server resp.	18.6 s	1123 ms	1156 ms	2058 ms	1183 ms	1321 ms	≈ 7100 ms	5677 ms	3621 ms
	Query size	14 KB	512 KB	448 KB	1339 KB	1486 KB	2254 KB	≈ 710 KB	140 KB	1184 KB
	Response size	20 KB	224 KB	12 KB	3008 KB	12 KB	120 KB	≈ 260 KB	26 KB	26 KB
	Throughput	440 MB/s	7295 MB/s	7087 MB/s	3981 MB/s	6925 MB/s	6201 MB/s	≈1154 MB/s	1443 MB/s	2262 MB/s
	Client comp.	90 ms	1814 ms	5039 ms	32.3 s	5103 ms	13.3 s	NA	7.7 ms	39.6 ms

Table 4: Compared with current state-of-the-art FHE-based PIR protocols. † Here we benchmark the performance of Simple/Double PIR from the YPIR library [48], which shows slightly improved performance over the implementations in the original paper [32, 33] and HintlessPIR [41, 42]. For 255 MB, 1 GB and 8 GB databases in HintlessPIR, we use database matrices of sizes 16384×16384, 32768×32768 and 32768×262144, respectively, with each entry being 8 bits. The clients of HintlessPIR can access a column of entries from the database matrix, so here the record size is 32 KB for an 8 GB database. "Server preproc." stands for the server preprocessing time. "Server resp." stands for the server response time. "Client comp." stands for the sum of query generation time and recovery time. Record sizes over 8 bits are unsupported for YPIR. We take the benchmarks of $2^{22}\times$ 256B (1 GB) and $2^{18}\times$ 32KB (8 GB) reported in Figure 1, 2 and 4 of WhisPIR's paper [20], where NA indicates that the concrete result is not publicly available.

throughput can approach that of Simple/Double PIR, albeit at the cost of reasonably larger online communication.

Compared with HintlessPIR, our stateful protocol demonstrates higher throughput when managing smaller databases, e.g., 256 MB. However, as the database size increases to 8 GB, HintlessPIR achieves a throughput of 3981 MB/s, which is faster than ours. Our stateless variant performs better, outperforming the HintlessPIR in both throughput and communication when the database is equal to

or smaller than 1 GB. Another disadvantage of HintlessPIR is that the rate is not good, where rate is the ratio of plaintext size to response size. This becomes particularly problematic when retrieving very large records, such as movies [46], where the response size may be substantially larger than that seen in other protocols. YPIR achieves the highest throughput when the records are 1 to 8 bits. The only limitation of YPIR is that its record size is somewhat small, which introduces some limitations in many application scenarios. Menon and Wu [49] also introduce another variant called YPIR + SP, which packs LWE ciphertexts in Simple PIR instead of Double PIR. YPIR + SP can support large records, but its throughput is not as good as YPIR's. Another disadvantage of YPIR + SP (which also includes HintlessPIR and YPIR) is the relatively high computational load on the client, which is not conducive to applications that require low latency on limited client resources.

Compared with WhisPIR. WhisPIR is a stateless protocol characterized by low communication overhead and good throughput. As a ring-based protocol, WhisPIR supports records of varying sizes, making it adaptable to most applications. Compared to Spiral, its only disadvantages are a slightly lower rate and marginally higher query size, but it offers better throughput and is fully stateless, which is particularly beneficial in scenarios with a large number of clients.

The implementation of WhisPIR is not publicly available at the time of this writing. It appears that communication and computation achieve a good tradeoff when the parameter is set to 16 chunks, so we simply take this benchmark reported in Figure 1, 2 and 4 of their paper [20]. Our stateful protocol has a slightly higher throughput and rate than WhisPIR but the server in our protocol have to store per-client storage. Taking the performance of 1 GB database as an example, our throughput is 1278 MB/s, slightly better than WhisPIR's 1024 MB/s. Moreover, our stateless protocol achieves a throughput of 2103 MB/s, approximately twice that of WhisPIR.

7 ACKNOLEWGEMENT

The authors would like to thank anonymous reviewers for their insightful comments that significantly help improve the presentation. Ming Luo and Han Wang are supported by the National Key R&D Program of China under Grant 2020YFA0712303, State Key Laboratory of Information Security under Grant TC20221013042, and the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDB0690200. Feng-Hao Liu is supported by NSF CNS-2402031.

REFERENCES

[1] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. 2021. Addra: Metadata-private voice communication over fully untrusted infrastructure. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*, Angela Demke Brown and Jay R. Lorch (Eds.). USENIX Association. https://www.usenix.org/conference/osdi21/presentation/ahmad

[2] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. 2021. Communication-Computation Trade-offs in PIR, See [6], 1811–1828.

[3] Jacob Alperin-Sheriff and Chris Peikert. 2014. Faster Bootstrapping with Polynomial Error, See [25], 297–314. https://doi.org/10.1007/978-3-662-44371-2_17

[4] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 962–979. https://doi.org/10.1109/SP.2018.00062

[5] Sebastian Angel and Srinath T. V. Setty. 2016. Unobservable Communication over Fully Untrusted Infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, Kimberly Keeton and Timothy Roscoe (Eds.). USENIX Association, 551–569. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/angel

[6] Michael Bailey and Rachel Greenstadt (Eds.). 2021. *USENIX Security 2021*. USENIX Association.

[7] Amos Beimel, Yuval Ishai, and Tal Malkin. 2004. Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing. *Journal of Cryptology* 17, 2 (March 2004), 125–151. https://doi.org/10.1007/s00145-004-0134-y

[8] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. 2016. Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 1006–1018. https://doi.org/10.1145/2976749.2978425

[9] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 353–367. https://doi.org/10.1109/EuroSP.2018.00032

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, 309–325. https://doi.org/10.1145/2090236.2090262

[11] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *52nd FOCS*, Rafail Ostrovsky (Ed.). IEEE Computer Society Press, 97–106. https://doi.org/10.1109/FOCS.2011.12

[12] Ran Canetti, Justin Holmgren, and Silas Richelson. 2017. Towards Doubly Efficient Private Information Retrieval. In *TCC 2017, Part II (LNCS)*, Yael Kalai and Leonid Reyzin (Eds.), Vol. 10678. Springer, Cham, 694–726. https://doi.org/10.1007/978-3-319-70503-3_23

[13] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2021. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In *ACNS 21International Conference on Applied Cryptography and Network Security, Part I (LNCS)*, Kazue Sako and Nils Ole Tippenhauer (Eds.), Vol. 12726. Springer, Cham, 460–479. https://doi.org/10.1007/978-3-030-78372-3_18

[14] Hao Chen, Kim Laine, and Rachel Player. 2017. Simple Encrypted Arithmetic Library - SEAL v2.1. In *FC 2017 Workshops (LNCS)*, Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y. A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson (Eds.), Vol. 10323. Springer, Cham, 3–18. https://doi.org/10.1007/978-3-319-70278-0_1

[15] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *ASIACRYPT 2016, Part I (LNCS)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.), Vol. 10031. Springer, Berlin, Heidelberg, 3–33. https://doi.org/10.1007/978-3-662-53887-6_1

[16] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. 1995. Private Information Retrieval. In *36th FOCS*. IEEE Computer Society Press, 41–50. https://doi.org/10.1109/SFCS.1995.492461

[17] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. 2022. Single-Server Private Information Retrieval with Sublinear Amortized Time. In *EUROCRYPT 2022, Part II (LNCS)*, Orr Dunkelman and Stefan Dziembowski (Eds.), Vol. 13276. Springer, Cham, 3–33. https://doi.org/10.1007/978-3-031-07085-3_1

[18] Henry Corrigan-Gibbs and Dmitry Kogan. 2020. Private Information Retrieval with Sublinear Online Time. In *EUROCRYPT 2020, Part I (LNCS)*, Anne Canteaut and Yuval Ishai (Eds.), Vol. 12105. Springer, Cham, 44–75. https://doi.org/10.1007/978-3-030-45721-1_3

[19] Alex Davidson, Gonçalo Pestana, and Sofia Celi. 2023. FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval. *Proc. Priv. Enhancing Technol.* 2023, 1 (2023), 365–383. https://doi.org/10.56553/popets-2023-0022

[20] Leo de Castro, Kevin Lewi, and Edward Suh. 2024. WhisPIR: Stateless Private Information Retrieval with Low Communication. Cryptology ePrint Archive, Paper 2024/266. https://eprint.iacr.org/2024/266 https://eprint.iacr.org/2024/266.

[21] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *EUROCRYPT 2015, Part I (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9056. Springer, Berlin, Heidelberg, 617–640. https://doi.org/10.1007/978-3-662-46800-5_24

[22] Zeev Dvir and Sivakanth Gopi. 2016. 2-Server PIR with Subpolynomial Communication. *J. ACM* 63, 4 (2016), 39:1–39:15. https://doi.org/10.1145/2968443

[23] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144 https://eprint.iacr.org/2012/144.

[24] Eric Fung, Georgios Kellaris, and Dimitris Papadias. 2015. Combining Differential Privacy and PIR for Efficient Strong Location Privacy. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings (Lecture Notes in Computer Science)*, Christophe

Claramunt, Markus Schneider, Raymond Chi-Wing Wong, Li Xiong, Woong-Kee Loh, Cyrus Shahabi, and Ki-Joune Li (Eds.), Vol. 9239. Springer, 295–312. https://doi.org/10.1007/978-3-319-22363-6_16

[25] Juan A. Garay and Rosario Gennaro (Eds.). 2014. *CRYPTO 2014, Part I*. LNCS, Vol. 8616. Springer, Berlin, Heidelberg.

[26] Craig Gentry and Shai Halevi. 2019. Compressible FHE with Applications to PIR. In *TCC 2019, Part II (LNCS)*, Dennis Hofheinz and Alon Rosen (Eds.), Vol. 11892. Springer, Cham, 438–464. https://doi.org/10.1007/978-3-030-36033-7_17

[27] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Fully Homomorphic Encryption with Polylog Overhead. In *EUROCRYPT 2012 (LNCS)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, Berlin, Heidelberg, 465–482. https://doi.org/10.1007/978-3-642-29011-4_28

[28] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *CRYPTO 2013, Part I (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8042. Springer, Berlin, Heidelberg, 75–92. https://doi.org/10.1007/978-3-642-40041-4_5

[29] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib, See [25], 554–571. https://doi.org/10.1007/978-3-662-44371-2_31

[30] Shai Halevi and Victor Shoup. 2018. Faster Homomorphic Linear Transformations in HElib. In *CRYPTO 2018, Part I (LNCS)*, Hovav Shacham and Alexandra Boldyreva (Eds.), Vol. 10991. Springer, Cham, 93–120. https://doi.org/10.1007/978-3-319-96884-1_4

[31] Ryan Henry. 2016. Polynomial Batch Codes for Efficient IT-PIR. *Proc. Priv. Enhancing Technol.* 2016, 4 (2016), 202–218. https://doi.org/10.1515/popets-2016-0036

[32] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3889–3905.

[33] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2024. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. https://github.com/ahenzinger/simplepir/commit/e9020b

[34] Wen jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. 2021. PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1057–1073. https://doi.org/10.1109/SP40001.2021.00043

[35] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. 2021. Revisiting Homomorphic Encryption Schemes for Finite Fields. In *ASIACRYPT 2021, Part III (LNCS)*, Mehdi Tibouchi and Huaxiong Wang (Eds.), Vol. 13092. Springer, Cham, 608–639. https://doi.org/10.1007/978-3-030-92078-4_21

[36] Dmitry Kogan and Henry Corrigan-Gibbs. 2021. Private Blocklist Lookups with Checklist, See [6], 875–892.

[37] Eyal Kushilevitz and Rafail Ostrovsky. 1997. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *38th FOCS*. IEEE Computer Society Press, 364–373. https://doi.org/10.1109/SFCS.1997.646125

[38] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. 2016. Riffle: An Efficient Communication System With Strong Anonymity. *Proc. Priv. Enhancing Technol.* 2016, 2 (2016), 115–134. https://doi.org/10.1515/popets-2016-0008

[39] Arthur Lazzaretti and Charalampos Papamanthou. 2022. Near-Optimal Private Information Retrieval with Preprocessing. Cryptology ePrint Archive, Paper 2022/830. https://eprint.iacr.org/2022/830 https://eprint.iacr.org/2022/830.

[40] Arthur Lazzaretti and Charalampos Papamanthou. 2023. TreePIR: Sublinear-Time and Polylog-Bandwidth Private Information Retrieval from DDH. Cryptology ePrint Archive, Paper 2023/204. https://eprint.iacr.org/2023/204 https://eprint.iacr.org/2023/204.

[41] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz-Wu. 2023. Hintless Single-Server Private Information Retrieval. Cryptology ePrint Archive, Report 2023/1733. https://eprint.iacr.org/2023/1733

[42] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz-Wu. 2023. Hintless Single-Server Private Information Retrieval. https://github.com/google/hintless_pir/commit/4be2ae

[43] Ming Luo, Feng-Hao Liu, and Han Wang. 2024. https://github.com/mmingluo/kspir.

[44] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT 2010 (LNCS)*, Henri Gilbert (Ed.), Vol. 6110. Springer, Berlin, Heidelberg, 1–23. https://doi.org/10.1007/978-3-642-13190-5_1

[45] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR : Private Information Retrieval for Everyone. *Proc. Priv. Enhancing Technol.* 2016, 2 (2016), 155–174. https://doi.org/10.1515/popets-2016-0010

[46] Samir Jordan Menon and David J. Wu. 2022. SPIRAL: Fast, High-Rate Single-Server PIR via FHE Composition. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 930–947. https://doi.org/10.1109/SP46214.2022.9833700

[47] Samir Jordan Menon and David J. Wu. 2023. SPIRAL: Fast, High-Rate Single-Server PIR via FHE Composition. https://github.com/menonsamir/spiral/commit/361ee4

[48] Samir Jordan Menon and David J. Wu. 2024. YPIR: High-Throughput Single-Server PIR with Silent Preprocessing. https://github.com/menonsamir/ypir/commit/8701cc

[49] Samir Jordan Menon and David J. Wu. 2024. YPIR: High-Throughput Single-Server PIR with Silent Preprocessing. Cryptology ePrint Archive, Paper 2024/270. https://eprint.iacr.org/2024/270 https://eprint.iacr.org/2024/270.

[50] Daniele Micciancio and Jessica Sorrell. 2018. Ring Packing and Amortized FHEW Bootstrapping. In *ICALP 2018 (LIPIcs)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.), Vol. 107. Schloss Dagstuhl, 100:1–100:14. https://doi.org/10.4230/LIPIcs.ICALP.2018.100

[51] Prateek Mittal, Femi G. Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. 2011. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *USENIX Security 2011*. USENIX Association.

[52] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response Efficient Single-Server PIR. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 2292–2306. https://doi.org/10.1145/3460120.3485381

[53] Muhammad Haris Mughees and Ling Ren. 2023. Vectorized Batch Private Information Retrieval. In *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 437–452. https://doi.org/10.1109/SP46215.2023.10179329

[54] Jeongeun Park and Mehdi Tibouchi. 2020. SHECS-PIR: Somewhat Homomorphic Encryption-Based Compact and Scalable Private Information Retrieval. In *ESORICS 2020, Part II (LNCS)*, Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider (Eds.), Vol. 12309. Springer, Cham, 86–106. https://doi.org/10.1007/978-3-030-59013-0_5

[55] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. 2018. Private Stateful Information Retrieval. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 1002–1019. https://doi.org/10.1145/3243734.3243821

[56] Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Don't be Dense: Efficient Keyword PIR for Sparse Databases. Cryptology ePrint Archive, Paper 2023/466. https://eprint.iacr.org/2023/466 https://eprint.iacr.org/2023/466.

[57] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A Framework for Efficient and Composable Oblivious Transfer. In *CRYPTO 2008 (LNCS)*, David Wagner (Ed.), Vol. 5157. Springer, Berlin, Heidelberg, 554–571. https://doi.org/10.1007/978-3-540-85174-5_31

[58] Giuseppe Persiano and Kevin Yeo. 2022. Limits of Preprocessing for Single-Server PIR. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, Joseph (Seffi) Naor and Niv Buchbinder (Eds.). SIAM, 2522–2548. https://doi.org/10.1137/1.9781611977073.99

[59] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *37th ACM STOC*, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, 84–93. https://doi.org/10.1145/1060590.1060603

[60] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce M. Maggs. 2021. Puncturable Pseudorandom Sets and Private Information Retrieval with Near-Optimal Online Bandwidth and Time. In *CRYPTO 2021, Part IV (LNCS)*, Tal Malkin and Chris Peikert (Eds.), Vol. 12828. Springer, Cham, Virtual Event, 641–669. https://doi.org/10.1007/978-3-030-84259-8_22

[61] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. 2020. Epione: Lightweight Contact Tracing with Strong Privacy. *IEEE Data Eng. Bull.* 43, 2 (2020), 95–107. http://sites.computer.org/debull/A20june/p95.pdf

[62] David J. Wu, Joe Zimmerman, Jérémy Planul, and John C. Mitchell. 2016. Privacy-Preserving Shortest Path Computation. In *NDSS 2016*. The Internet Society. https://doi.org/10.14722/ndss.2016.23052

[63] Kevin Yeo. 2023. Lower Bounds for (Batch) PIR with Private Preprocessing. In *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I (Lecture Notes in Computer Science)*, Carmit Hazay and Martijn Stam (Eds.), Vol. 14004. Springer, 518–550. https://doi.org/10.1007/978-3-031-30545-0_18

[64] Mingxun Zhou, Wei-Kai Lin, Yiannis Tselekounis, and Elaine Shi. 2023. Optimal Single-Server Private Information Retrieval. In *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part I (Lecture Notes in Computer Science)*, Carmit Hazay and Martijn Stam (Eds.), Vol. 14004. Springer, 395–425. https://doi.org/10.1007/978-3-031-30545-0_14

[65] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. 2023. Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation. Cryptology ePrint Archive, Report 2023/452. https://eprint.iacr.org/2023/452