Information-Theoretic Multi-Server PIR with Global Preprocessing

Ashrujit Ghoshal
CMU
aghoshal@andrew.cmu.edu

Baitian Li
Tsinghua IIIS and Columbia
lbt21@mails.tsinghua.edu.cn

Yaohua Ma
Tsinghua IIIS and CMU
yaohuam@andrew.cmu.edu

Chenxin Dai
Tsinghua IIIS and CMU
chenxind@andrew.cmu.edu

Elaine Shi*
CMU
runting@gmail.com

Abstract

We propose a new unified framework to construct multi-server, information-theoretic Private Information Retrieval (PIR) schemes that leverage global preprocesing to achieve sublinear computation per query. Despite a couple earlier attempts, our understanding of PIR schemes in the global preprocessing model remains limited, and so far, we only know a few sparse points in the broad design space. With our new unified framework, we can generalize the results of Beimel, Ishai, and Malkin to broader parameter regimes, thus enabling a tradeoff between bandwidth and computation. Specifically, for any constant $S > 1$, we can get an S-server scheme whose bandwidth consumption is as small as $n^{1/(S+1)+\epsilon}$ while achieving computation in the n^{δ} regime for some constant $\delta \in (0, 1)$. Moreover, we can get a scheme with polylogarithmic bandwidth and computation, requiring only polylogarithmic number of servers.

1 Introduction

Private Information Retrieval (PIR), originally proposed by Chor et al. [CGKS95], allows a client to retrieve an entry from a public database stored on one or more server(s), without leaking its query to any individual server. PIR promises numerous applications such as private DNS [Fea, obl, SCV+21], privately checking whether one's password is in some leaked password database [hav, DRRT18], private contact discovery [sig], private web search [HDCG+23], and so on. In classical PIR schemes [CGKS95, Cha04, GR05, CMS99, CG97, KO97, Lip09, OS07, Gas04, BFG03, SC07, OG11, MCG+08, MG07, HHCG+23, MW22], the servers store the original database and there is no preprocessing. Unfortunately, Beimel, Ishai, and Malkin [BIM00] proved any classical PIR scheme (without preprocessing) suffers from a fundamental limitation, that is, every query must incur a linear (in the database size) amount of server computation. Intuitively, if there is some entry that the server does not look at to answer a client's query, then the client cannot be interested in that entry.

To scale PIR to large datasets, Beimel et al. [BIM00] introduced a new preprocessing model for PIR, and showed that with preprocessing, we can overcome the linear server computation barrier. In the *global* preprocessing model proposed by Beimel et al. [BIM00], the server computes and stores an encoded version of the database which can be polynomial in size, and the same preprocessing is shared across all clients. Subsequent works have also considered a *client-specific* preprocessing model [CK20, CHK22, ZLTS23, LP23, LP22, SACM21, ZPSZ24, GZS24] where each client performs a separate preprocessing with the server (also called the subscription phase), and at the end of the preprocessing each client stores a hint that is related to the database. In comparison with client-specific preprocessing, the global preprocessing model enjoys some

*Author ordering is randomized.

advantages. First, the same preprocessing can be amortized to an unbounded number of clients. In other words, the total preprocessing work and total space consumption do not depend on the number of clients. Second, for a dynamically evolving database, PIR with global preprocessing can easily be made dynamic using the standard hierarchical data structure by Bentley and Saxe [BS80], whereas with client-specific preprocessing, *every* client would have to update its hint for each update to the database [KCG21, HPPY24], and the costs can be significant for fast-evolving databases.

In this work, we focus on information-theoretic PIR schemes in the global preprocessing model, which is exactly the model considered by Beimel et al. [BIM00]. We focus on the setting of two or more servers which is necessary for achieving information-theoretic security due to well-known lower bounds [DMO00]. Since we focus on the global preprocessing model, we are particularly interested in PIR schemes that achieve *sublinear* computation per query.

Status quo and open questions. Despite earlier attempts [BIM00, WY05], our existing understanding of information-theoretic PIR in the global preprocessing model is rather limited. As we discuss below, so far, we only know a few sparse points in the entire design space; and numerous open questions remain.

First, in the *2-server* setting, Beimel et al. [BIM00] and the subsequent work of Woodruff and Yekhanin [WY05] showed that with a polynomial amount of server space, we can get PIR schemes with $O(n^{1/3})$ bandwidth and $n/\mathsf{poly}\log n$ computation per query, where n denotes the size of the database. The obvious disadvantage is that the computation is only *slightly sublinear*, and it would be more desirable to get schemes whose computation is *significantly bounded away from linear*, e.g., in the n^δ regime for some constant $\delta \in (0, 1)$. For this regime, the only known result is a scheme also proposed by Beimel et al. [BIM00], who showed how to achieve $n^{1/2+\epsilon}$ cost per query both in terms of bandwidth and server computation. However, the parameter choices in this scheme are fixed and there does not seem to be any straightforward way to enable a more general tradeoff between the bandwidth and the computation overhead (see also Section 1.2). In particular, if we did not mind having linear server computation, then it is known how to achieve bandwidth as small as $n^{o(1)}$ [DG16]. Therefore, a natural open question is the following:

> Open question 1: *Can we have a PIR scheme in the global preprocessing model with n^δ server computation for some $\delta \in (0, 1)$, and moreover with bandwidth asymptotically smaller than $n^{1/2}$?*

More generally, in the S-server setting, the only known results are due to Beimel et al. [BIM00]. They showed two results for this setting: 1) a scheme with $O(n^{1/(2S-1)})$ bandwidth and $O(n/\log^{2S-2} n)$ computation per query; and 2) a scheme with $O(n^{1/S+\epsilon})$ bandwidth and computation per query for an arbitrarily small constant $\epsilon > 0$. Therefore, our understanding of the S-server setting is limited in the following sense. First, both of the aforementioned results *require the number of servers S to be a constant*; otherwise the server space will be super-polynomial. Recall that if we did not mind having linear server computation, then indeed it is known how to get an $O(\log n)$-server scheme with $O(\log^2 n \log\log n)$ bandwidth in the classical setting [CGKS95, BF90]. Unfortunately, so far we do not have an analogous result in the preprocessing setting with sublinear server computation. In fact, for any super-constant S, we do not know any information-theoretic construction that achieves sublinear computation per query[1]. Therefore, a natural open question is:

> Open question 2: *Can we get a non-trivial scheme with sublinear computation under a super-constant number of servers?*

Moreover, Beimel et al.'s results require a fixed parametrization and there does not seem to be a straightforward way to enable a more general tradeoff between the bandwidth and computation overhead. Therefore, another open question is:

[1]Except for the trivial approach of ignoring all but $O(1)$ number of servers — however, this trivial approach cannot utilize the additional servers to further reduce bandwidth.

Table 1: 2-server information-theoretic PIR

Scheme	Server Compute	Bandwidth	Server Space
[DG16]	$\geq n$	$n^{o(1)}$	0
[BIM00]	$n^{1/2+\epsilon}$	$n^{1/2+\epsilon}$	$n^{1+\epsilon'}$
[BIM00]	$O(n/\log^2 n)$	$O(n^{1/3})$	$O(n^2)$
[WY05]	$n/\text{poly}\log n$	$O(n^{1/3})$	$\text{poly}(n)$
Section 4 ($\forall 1/3 \leq \alpha \leq 1/2$)	$n^{1-\alpha+\epsilon}$	$n^{\alpha+\epsilon}$	$\text{poly}(n)$

Table 2: S-server information-theoretic PIR

Scheme	Server Compute	Bandwidth	Server Space	Assumption
[BIM00]	$n^{1/S+\epsilon}$	$n^{1/S+\epsilon}$	$\text{poly}(n)$	$S = O(1)$
[BIM00]	$O(n/\log^{2S-2} n)$	$O(n^{1/(2S-1)})$	$\text{poly}(n)$	$S = O(1)$
Section 4 ($\forall \frac{1}{S+1} \leq \alpha \leq \frac{1}{S}$)	$n^{1-(S-1)\alpha+\epsilon}$	$n^{\alpha+\epsilon}$	$\text{poly}(n)$	$S = O(1)$
Appendix B	$\text{poly}\log(n)$	$\text{poly}\log(n)$	$\text{poly}(n)$	$S = \text{poly}\log n$

Open question 3: *Can we get an S-server information-theoretic PIR scheme with per-query bandwidth asymptotically less than $O(n^{1/S})$ and computation in the n^δ regime for some constant $\delta \in (0,1)$?*

1.1 Our Results and Contributions

In this work, we propose a new, unified framework for constructing multi-server information-theoretic PIR in the global preprocessing model. Using our new framework, we can generalize the results of Beimel et al. [BIM00] to broader parameter regimes, thus providing an affirmative answer to all the open questions posed earlier.

We now summarize our results and contributions.

Theorem 1.1. *Suppose that the number of servers S is a constant. For any $1/(S+1) \leq \alpha \leq 1/S$ and an arbitrarily small $\epsilon > 0$, there exists an information-theoretic S-server preprocessing PIR scheme with $n^{\alpha+\epsilon}$ bandwidth and client computation, and $n^{1-(S-1)\alpha+\epsilon}$ server computation per query, assuming* $\text{poly}(n)$ *amount of server space.*

Note that if we take $\alpha = 1/S$, our result is the same as that of Beimel et al. [BIM00]. However, our scheme admits a broader range of parameter choices which allow a tradeoff between the bandwidth and computation overhead. Specifically, we can make the bandwidth as small as $n^{1/(S+1)+\epsilon}$ while still keeping the computation in the n^δ regime.

For the special case when $S = 2$, the above Theorem 1.1 immediately gives rise to the following corollary.

Corollary 1.2. *For any $1/3 \leq \alpha \leq 1/2$, for an arbitrarily small $\epsilon \in (0,1)$, there exists an information-theoretic 2-server preprocessing PIR scheme with $n^{\alpha+\epsilon}$ bandwidth and client computation, and $n^{1-\alpha+\epsilon}$ server computation per query, assuming* $\text{poly}(n)$ *amount of server space.*

As a special case, by taking $\alpha = 1/3$, we get a scheme with $n^{1/3+\epsilon}$ and $n^{2/3+\epsilon}$ computation per query. Due to the lower bound of Razborov and Yakhanin [RY06], we know that the $n^{1/3+\epsilon}$ bandwidth is (nearly) optimal for a natural class of bilinear and group-based 2-server PIR schemes. So far, with the exception of Dvir and Gopi [DG16], all known 2-server PIR schemes in the classical or the global preprocessing models adopt this natural paradigm, including our new constructions. This provides some evidence that further improving the bandwidth would require significantly different techniques.

Theorem 1.1 is for the setting when $S = O(1)$. We also present a new result for the setting when $S = \mathsf{poly}\log n$, showing that one can achieve polylogarithmic computation and bandwidth per query. We state this result in the following theorem:

Theorem 1.3. *There exists an* $\mathsf{poly}\log n$*-server information-theoretic PIR scheme with* $\mathsf{poly}\log n$ *bandwidth and computation per query.*

We compare our results with prior work in Table 1 and Table 2. We present our technical highlights in Section 1.2 and provide a more detailed overview in Section 2.

1.2 Technical Highlight

To get a PIR scheme with sublinear computation in the global preprocessing setting, Beimel et al. [BIM00] first constructs a *classical* PIR scheme whose query length is $O(\log n)$. Since the query is short, each server can precompute and store the answers to all possible queries, requiring only $\mathsf{poly}(n)$ preprocessing time and $\mathsf{poly}(n)$ server space. During the online query, the server can simply look up the answer. Therefore, this approach fundamentally restricts us to getting a scheme where the server computation equals the download bandwidth (which dominates the total bandwidth), and does not admit a more fine-grained tradeoff between the bandwidth and computation overheads.

Blueprint. Our new framework works as follows. First, we suggest an improvement of the scheme by Woodruff and Yekhanin [WY05]. Specifically, the query length of Woodruff and Yekhanin [WY05]'s scheme is $\omega(\log n)$. For example, the query length of their 2-server scheme is at least $n^{1/3}$. We show an improved construction where the query length is $O(\log n)$. This way, the server can simply precompute and store answers to all possible queries just like Beimel et al. [BIM00], which immediately allows us to match the result of Beimel et al. [BIM00], that is, an S-server scheme with $O(n^{1/S+\epsilon})$ bandwidth and computation per query. Moreover, the advantage of our new framework is that it admits a balancing trick described by Woodruff and Yekhanin [WY05]. Specifically, without the balancing, the upload bandwidth (i.e., query length) is $O(\log n)$, but the download bandwidth is $O(n^{1/S+\epsilon})$; therefore, the two are unbalanced. The balancing trick allows us to reduce the download bandwidth by making the upload bandwidth and the server computation bigger, which enables a general tradeoff between bandwidth and computation. As a special point of interest, if we aim to minimize bandwidth while keeping the server computation in the n^δ regime, we can make both the upload and download bandwidth $O(n^{1/(S+1)+\epsilon})$ at the price of having $O(n^{2/(S+1)+\epsilon})$ server computation.

Novel techniques. For a technical perspective, our most novel idea is how to reduce the query length of Woodruff and Yekhanin [WY05] to $O(\log n)$. Woodruff and Yekhanin represent the database as a constant-degree polynomial with $m = O(n^{1/3})$ variables over a constant-sized field $\mathbb{F}_q$ (for the case when $S = 2$). Moreover, the query to the server is a length-m vector in $\mathbb{F}_q^m$.

Instead, we want to represent the database as a polynomial with only $m = O(\log n)$ variables over $\mathbb{F}_q$ — for this to be possible, we need to increase the degree of the polynomial to $d = O(\log n)$. Unfortunately, Woodruff and Yekhanin's techniques only work if the field size is larger than the degree of the polynomial d. Otherwise, the higher-order derivatives computed by the client during reconstruction will degenerate to

0, and their reconstruction algorithm would thus fail to work. Unfortunately, if we increase the field size to as large as d, the query length would no longer be $O(\log n)$.

We suggest a new reconstruction algorithm compatible with Woodruff and Yekhanin's framework, such that we can still work with a constant-sized field even when the degree of the polynomial is larger. To achieve this, our main novel idea is to replace the derivative calculation during reconstruction with *Hasse derivatives* instead. We show that using Hasse derivatives allows us to overcome the problem of the higher-order derivatives vanishing to 0, and reconstruction is possible by solving a linear system represented by a matrix whose determinant is non-zero.

We refer the reader to Section 2 for a more detailed explanation of the limitations of Woodruff and Yekhanin's techniques, as well as how we use Hasse derivatives during reconstruction to overcome these limitations. We defer additional related work to Appendix D.

2 Informal Overview

2.1 Background: The Framework of Woodruff and Yekhanin

We first give some background on the 2-server PIR framework of Woodruff and Yekhanin [WY05]. Recall that Woodruff and Yekhanin's scheme achieves $O(n^{1/3})$ bandwidth. Howerver, for clarity, we will first describe a simplified version of their scheme that achieves $O(n^{1/2})$ bandwidth.

Notations. We define A_k to be the set of all binary vectors of length m and Hamming weight exactly k:

$$A_k = \{\vec{a} \in \{0,1\}^m : \mathsf{wt}(\vec{a}) = k\}$$

where $\mathsf{wt}(\vec{a}) = a_1 + \ldots + a_m$ denotes the Hamming weight of the vector $\vec{a}$. Let $A_{\leq k} := A_0 \cup A_1 \ldots \cup A_k$.

Given $\vec{a} := (a_1, \ldots, a_m) \in \{0,1\}^m$, and a polynomial F, we define the partial derivative operator $\boldsymbol{\partial}^{\vec{a}}$ as:

$$\boldsymbol{\partial}^{\vec{a}} \circ F := \frac{\partial^{\mathsf{wt}(\vec{a})} F}{\partial X_1^{a_1} \ldots \partial X_m^{a_m}}$$

Henceforth, given a vector $\vec{X} := (X_1, \ldots, X_m)$ of variables and a vector $\vec{a} := (a_1, \ldots, a_m)$ of exponents, we use the following vector exponentiation notation:

$$\vec{X}^{\vec{a}} := \prod_k X_k^{a_k}$$

Polynomial encoding of the database. Let $\mathbb{F}_q$ denote some finite field of order q. The idea of Woodruff and Yekhanin [WY05] is to encode the database $\mathsf{DB} \in \{0,1\}^n$ as an m-variable polynomial in $\mathbb{F}_q[X_1, \ldots, X_m]$, of homogeneous degree d and individual degree at most 1. In other words, all monomials are of degree d and in each monomial, every variable has degree at most 1. To achieve this, we can define some injective indexing function $E : [n] \to \mathbb{F}_q^m$, and we interpolate a polynomial $F \in \mathbb{F}_q[X_1, \ldots, X_m]$ such that for $i \in [n]$, $F(E(i)) = \mathsf{DB}[i]$. For the interpolation to be successful, we need that $\binom{m}{d} \geq n$. We additionally assume that the field size $q > d$ and the reason for this will become clear later.

Protocol. To retrieve $\mathsf{DB}[i]$, the client relies on the following protocol to find out the evaluation of the polynomial at $E(i)$.

1. For the queried index $i \in [n]$, let $\vec{u} = E(i)$. For each $s \in \{0,1\}$, the client randomly picks $\vec{v} \in \mathbb{F}_q^m$, and sends $\vec{z}_s := \vec{u} + (-1)^s \vec{v}$ to server s.

2. Server $s \in \{0,1\}$ receives the vector $\vec{z}_s \in \mathbb{F}_q^m$. The server computes the original polynomial F evaluated at $\vec{z}_s$, as well as all derivatives of F of order at most $\lfloor d/2 \rfloor$ also evaluated at $\vec{z}_s$, and returns the results to the client. More formally, for each $\vec{a} \in A_{\leq k}$, the server computes $\boldsymbol{\partial}^{\vec{a}} \circ F(\vec{z}_s)$ and sends the result back to the client.

3. The client reconstructs $\mathsf{DB}[i] = F(\vec{u})$ from the answers it obtains from the two servers. We explain how the reconstruction works below.

Reconstruction algorithm. Consider the univariate polynomial $f(\lambda) = F(\vec{u} + \lambda\vec{v})$ and $g(\lambda) = f(\lambda) + f(-\lambda)$. Observe that g must be a degree-d polynomial in $\mathbb{F}_q[\lambda]$, and $F(\vec{u}) = \frac{1}{2}g(0)$ gives the desired answer. Therefore, to compute $\mathsf{DB}[i] = F(\vec{u})$, it suffices to reconstruct the polynomial g. To achieve this, the client computes the sequence of derivatives $g(1), g^{(1)}(1), g^{(2)}(1), \ldots, g^{(\lfloor d/2 \rfloor)}(1)$ — these derivatives can be computed using the chain rule (Lemma 3.1) from the server's responses $\{\boldsymbol{\partial}^{\vec{a}} \circ F(\vec{z}_s)\}_{s \in \{0,1\}, \vec{a} \in A_{\leq \lfloor d/2 \rfloor}}$. Here, we need the condition that $q > d$ in order for the derivatives $g^{(1)}, \ldots, g^{(\lfloor d/2 \rfloor)}$ to not degenerate to 0.

Finally, given $g(1), g^{(1)}(1), g^{(2)}(1), \ldots, g^{(\lfloor d/2 \rfloor)}(1)$, the client can recover the polynomial g by solving a linear system. In particular, observe that the polynomial $g(\lambda)$ only has even-degree terms, that is, it can be expressed in the form $g(\lambda) = \sum_{j \in \lfloor d/2 \rfloor} c_j \cdot \lambda^{2j}$. Each derivative $g^{(i)}(1)$ imposes a linear constraint on the coefficients $\{c_j\}_{j \in [\lfloor d/2 \rfloor]}$. One can show that all $\lfloor d/2 \rfloor$ linear equations are linearly independent so the reconstruction is possible through the standard Gaussian elimination algorithm.

Bandwidth. Woodruff and Yekhanin [WY05] choose the parameters as follows. Consider the special case where $d = 3$. In this case, $m = O(n^{1/3})$, and the server's response contains the original polynomial F and all first-order derivatives $\frac{\partial F}{\partial X_1}, \ldots, \frac{\partial F}{\partial X_m}$ evaluated at the specified point. Thus the response size is $O(n^{1/3})$. Therefore, the bandwidth is bounded by $O(n^{1/3})$.

Naïve "precompute-all" approach for achieving sublinear server computation. The remaining question is how to make the server computation sublinear. One flawed idea is the following: for every polynomial in $\{\boldsymbol{\partial}^{\vec{a}} \circ F\}_{\vec{a} \in A_{\leq \lfloor d/2 \rfloor}}$, both servers precompute and store its evaluation at all q^m possible points. This way, each server only needs to perform a table lookup to evaluate each $\boldsymbol{\partial}^{\vec{a}} \circ F$ at the specified point, and thus the server computation per query would be upper bounded by the size of $A_{\leq \lfloor d/2 \rfloor}$. Unfortunately, given the requirement $\binom{m}{d} \geq n$ and $q > d$, one can show that q^m must be super-polynomial, i.e., precomputing at all possible points would require super-polynomial server space.

Instead of the naïve precompute-all approach, Woodruff and Yekhanin [WY05] adopt a different approach which partly relies on the Method of Four Russians[2] to achieve $n/\mathsf{poly}\log n$ server computation. However, since our new approach will not rely on this trick, we omit its full description.

2.2 Our New Idea: Warmup Scheme with $n^{1/2+\epsilon}$ Bandwidth

We want to give the naïve "precompute-all" approach another chance. In Woodruff and Yekhanin's framework [WY05], it would have been nice if we could use a field $\mathbb{F}_q$ of characterstic $q = O(1)$, and set the number of variables $m = O(\log n)$. This way, q^m is upper bounded by $\mathsf{poly}(n)$, and thus the "precompute-all" approach would require only polynomial amount of server space. Unfortunately, to have a small number of variables $m = O(\log n)$, we would need the total degree of the polynomial to be $d \geq \Theta(\log n)$ in order for the requirement $\binom{m}{d} \geq n$ to hold. However, recall that for the reconstruction algorithm to work, we would need that $q > d \geq \Theta(\log n)$ since otherwise the derivatives $g^{(1)}, \ldots, g^{(\lfloor d/2 \rfloor)}$ would all degenerate to 0. This contradicts the condition $q = O(1)$ that is needed for efficiency.

[2]Partitioning $[m]$ into $\log m$ sized chunks in [WY05] is inspired by the Method of Four Russians.

A new reconstruction algorithm using Hasse derivatives. Our idea is to use the same protocol of Woodruff and Yekhanin [WY05], but instead choose the number of variables $m = O(\log n)$ and choose $d = \theta m$ for some suitable constant $\theta \in (0, 1)$, such that $\binom{m}{d} \geq n$. Further, we will work with a field of small characteristic $q = 3$, such that q^m is polynomially bounded.

To make this idea work, our main contribution is to devise a new construction algorithm that is compatible with a small-characteristic field. Instead of using the derivatives $g^{(1)}, \ldots, g^{(\lfloor d/2 \rfloor)}$ to reconstruct g, we will use the *Hasse derivatives* of g instead.

Definition 2.1 (Hasse derivatives)**.** Let $\mathbb{F}[\lambda]$ be a polynomial ring over the field over the field $\mathbb{F}$. The r-th Hasse derivative of λ^m is defined as

$$\overline{\partial}^{(r)} \lambda^m = \begin{cases} \binom{m}{r} \cdot \lambda^{m-r} & \text{if } m \geq r \\ 0 & \text{o.w.} \end{cases}$$

For a degree-d polynomial $f(\lambda) = \sum_{k=0}^{d} c_k \cdot \lambda^k \in \mathbb{F}[\lambda]$, its r-th Hasse derivative is $\overline{\partial}^{(r)} f(\lambda) = \sum_{k=r}^{d} c_k \cdot \binom{k}{r} \lambda^{k-r}$. If the field $\mathbb{F}$ has characteristic 0, then $\overline{\partial}^{(r)} f(\lambda) = \frac{1}{r!} f^{(r)}(\lambda)$.

Suppose $f(\lambda) \in \mathbb{F}_q[\lambda]$ is a degree-d polynomial defined over a finite field $\mathbb{F}_q$, and suppose $d \geq r \geq q$. Unlike the normal r-th derivative of f which would always degenerate to 0; the r-th Hasse derivative of f does not necessarily degenerate to 0.

Using the Hasse derivatives, we will adopt a new reconstruction algorithm. As before, let $g(\lambda) = F(\vec{u} + \lambda \vec{v}) + F(\vec{u} - \lambda \vec{v}) \in \mathbb{F}_q[\lambda]$ be a univariate degree-d polynomial over $\mathbb{F}_q$. Now, instead of computing the normal derivatives $g(1), g^{(1)}(1), g^{(2)}(1), \ldots, g^{(\lfloor d/2 \rfloor)}(1)$, the client will instead compute the Hasse derivatives $g(1), \overline{\partial}^{(1)} g(1), \ldots, \overline{\partial}^{(\lfloor d/2 \rfloor)} g(1)$. The Hasse derivatives can be computed using the chain rule from the server's responses $\{\boldsymbol{\partial} \circ F(\vec{z}_s)\}_{s \in \{0,1\}, \boldsymbol{\partial} \in D_{\leq \lfloor d/2 \rfloor}}$ — see Lemma 3.2.

The remaining question is whether the client can reconstruct the polynomial g from these Hasse derivatives. Recall that g has only even-degree terms, that is, it can be expressed as $g(\lambda) = \sum_{k=0}^{\lfloor d/2 \rfloor} c_k \lambda^{2k}$. Thus, the Hasse derivatives $g(1), \overline{\partial}^{(1)} g(1), \ldots, \overline{\partial}^{(\lfloor d/2 \rfloor)} g(1)$ that the client knows define a system of linear equations over the coefficients $\{c_k\}_{k \in \{0, \ldots, \lfloor d/2 \rfloor\}}$:

$$\begin{cases} \sum_{k=0}^{\lfloor d/2 \rfloor} \binom{2k}{0} c_k & = \overline{\partial}^{(0)} g(1) \\ \sum_{k=0}^{\lfloor d/2 \rfloor} \binom{2k}{1} c_k & = \overline{\partial}^{(1)} g(1) \\ \ldots \\ \sum_{k=0}^{\lfloor d/2 \rfloor} \binom{2k}{l} c_k & = \overline{\partial}^{(l)} g(1) \\ \ldots \\ \sum_{k=0}^{\lfloor d/2 \rfloor} \binom{2k}{\lfloor d/2 \rfloor} c_k & = \overline{\partial}^{(\lfloor d/2 \rfloor)} g(1). \end{cases}$$

Observe that there are $\lfloor d/2 \rfloor + 1$ unknown variables $c_0, \ldots c_{\lfloor d/2 \rfloor}$, and $\lfloor d/2 \rfloor + 1$ linear equations. We prove that as long as $q \geq 3$, the matrix of coefficients has full rank, and thus the linear system can be solved using Gaussian elimination. The proof of this lemma is rather technical and we defer the full proof to Lemma 4.5.

Performance bounds. Given an arbitrarily small constrant $\theta \in (0, 1)$, suppose we choose $m = O(\log n)$, $d = \theta m$ such that $\binom{m}{d} \geq n$. Then, the size of the set $D_{\leq \lfloor d/2 \rfloor}$ is $n^{1/2+\epsilon}$ where $\epsilon \in (0, 1)$ is an arbitrarily small constant dependent on θ. Therefore, the bandwidth per query is $n^{1/2+\epsilon}$. The servers can precompute and store the answers at all possible points, and the space required is upper bounded by $q^m \cdot n^{1/2+\epsilon} = n^{\exp(O(1/\epsilon))} = \mathsf{poly}(n)$. This way, the server only needs to perform $n^{1/2+\epsilon}$ lookups to obtain the answers to the client, and thus the per-query server computation is also upper bounded by $n^{1/2+\epsilon}$. Since the Gaussian elimination computed by the client takes only $\mathsf{poly}(m) = \mathsf{poly}\log(n)$ time, the client computation is bounded by its bandwidth overhead, i.e., $n^{1/2+\epsilon}$.

2.3 Reducing the Bandwidth to $n^{1/3+\epsilon}$ with a Balancing Trick

In the above scheme, the upload bandwidth is only $m = O(\log n)$; however, the download bandwidth is $n^{1/2+\epsilon}$. Thus, the upload bandwidth and the download bandwidth are unbalanced. We can use a balancing trick described by Woodruff and Yekhanin [WY05] to reduce the bandwidth to $n^{1/3+\epsilon}$, at the cost of having $n^{2/3+\epsilon}$ computation per query.

The idea is to divide the database of n bits into $n^{1/3}$ blocks each of size $n^{2/3}$. We will treat each block as a separate database, and use an m-variate polynomial of *homogeneous degree* d to encode it; further, we require that d be *odd* which will be important for the unrelated blocks to cancel out during reconstruction. We choose $m = O(\log n)$ such that $\binom{m}{d} \geq n^{2/3}$. Let $F_0, \ldots, F_{n^{1/3}-1}$ be polynomials that encodes each of the $n^{1/3}$ blocks.

Now to fetch the database at some desired index, the client will prepare a PIR query for each of the $n^{1/3}$ instances; thus, the upload bandwidth is $O(n^{1/3})$. To get $n^{1/3+\epsilon}$ download bandwidth, we need to use an aggregation trick such that the server can aggregate the answers from all $n^{1/3}$ instances. When the client performs the reconstruction, all irrelevant instances will cancel out, and the answer from the relevant block (i.e., where the queried index belongs) will emerge.

Our 2-server protocol. More formally, the modified protocol works as follows.

- Let $i \in [n]$ be the queried index, let $r = \lfloor i/n^{1/3} \rfloor \in \{0, 1, \ldots, n^{1/3}-1\}$ be the block where i resides, and we will encode its offset within the block as a vector $\vec{u} := E((i \mod n^{1/3}) + 1) \in \mathbb{F}_q^m$.

- For every block $j \in \{0, 1, \ldots, n^{1/3}-1\}$, the client picks a random $\vec{v}_j \in \mathbb{F}_q^m$. It sends to server $s \in \{0, 1\}$ the vectors $\vec{z}_0, \ldots, \vec{z}_{n^{1/3}-1}$ where

$$\vec{z}_j := \begin{cases} (-1)^s \vec{v}_j & \text{if } j \neq r \\ \vec{u} + (-1)^s \vec{v}_r & \text{if } j = r \end{cases}$$

  Note that $\vec{u}$ which encodes the query is only added to the r-th vector.

- For each server $s \in \{0, 1\}$, on receiving $n^{1/3}$ vectors denoted $\{\vec{z}_j\}_{j \in \{0, \ldots, n^{1/3}-1\}}$, do the following: for each $\vec{a} \in A_{\leq \lfloor d/2 \rfloor}$, compute $\mathsf{ans}_{s,\vec{a}} := \sum_{j \in \{0,1,\ldots,n^{1/3}-1\}} \boldsymbol{\partial}^{\vec{a}} \circ F_j(\vec{z}_j) \cdot (-1)^{s \cdot \mathsf{wt}(\vec{a})}$ and send $\{\mathsf{ans}_{s,\vec{a}}\}_{\vec{a} \in A_{\leq \lfloor d/2 \rfloor}}$ back to the client.

- For each $\vec{a} \in A_{\leq \lfloor d/2 \rfloor}$, the client computes

$$\mathsf{ans}_{\vec{a}} := \sum_{s \in \{0,1\}} \mathsf{ans}_{s,\vec{a}} = \boldsymbol{\partial}^{\vec{a}} \circ F_r(\vec{u} + \vec{v}_r) + \boldsymbol{\partial}^{\vec{a}} \circ F_r(\vec{u} - \vec{v}_r) \cdot (-1)^{\mathsf{wt}(\vec{a})}$$

  In the above, notice that the contributions of all the irrelevant blocks $j \neq r$ cancel out. Specifically, since every F_j is a polynomial of odd homogeneous degree, it is not hard to verify that $\sum_{s \in \{0,1\}} \boldsymbol{\partial}^{\vec{a}} \circ F_j(\vec{z}_j) \cdot (-1)^{s \cdot \mathsf{wt}(\vec{a})} = 0$ for $j \neq r$.

  Therefore, define $g(\lambda) := F_r(\vec{u} + \lambda \vec{v}_r) + F_r(\vec{u} - \lambda \vec{v}_r)$, the client can now use the chain rule to compute $\overline{\partial}^{(0)} g(1), \overline{\partial}^{(1)} g(1), \ldots, \overline{\partial}^{(\lfloor d/2 \rfloor)} g(1)$. Specifically,

$$\overline{\partial}^{(k)} g(1) := \sum_{\vec{a} \in A_k} \left(\underbrace{\boldsymbol{\partial}^{\vec{a}} \circ F_r(\vec{u} + \vec{v}_r) + \boldsymbol{\partial}^{\vec{a}} \circ F_r(\vec{u} - \vec{v}_r) \cdot (-1)^{\mathsf{wt}(\vec{a})}}_{= \mathsf{ans}_{\vec{a}}} \right) \cdot \vec{v}_r^{\vec{a}}$$

From these derivatives it can reconstruct the polynomial g and compute $\mathsf{DB}[i] = \frac{1}{2}g(0)$.

In the above scheme, the client sends to the server $n^{1/3}$ vectors each of length $m = O(\log n)$. The server's response size is bounded by $O((n^{2/3})^{1/2+\epsilon}) = O(n^{1/3+\epsilon'})$. Therefore, the total bandwidth is bounded by $O(n^{1/3+\epsilon'})$. The server computation is bounded $O((n^{2/3})^{1/2+\epsilon}) = O(n^{1/3+\epsilon'})$. As before, the client computation is dominated by its bandwidth which is $O(n^{1/3+\epsilon'})$.

2.4 Extension to Multiple Servers

Constant number of servers. We can extend our ideas to S servers for any constant $S > 2$. Here we choose a field $\mathbb{F}_q$ where q is a prime and has an S-th root of unity denoted by ω. Here we divide n-bit database into $B = n^{1-\mu}$ blocks of size n^μ bits each. Like for the 2-server case will, we will use an m-variate polynomial of homogeneous degree d to encode it; we require that d is not a multiple of S which will be important for the unrelated blocks to cancel out during reconstruction. We choose $m = O(\log n)$ and $d = \theta m$ for $0 < \theta \leq 1/2$ such that $\binom{m}{d} \geq n^\mu$. We show that such a choice of d is possible in Lemma 3.4. Let $F_0, \ldots, F_{B-1}$ be polynomials that encodes each of the B blocks.

Now to fetch the database at some desired index, the client will prepare a PIR query for each of the B instances. We again use an aggregation trick, and set $\mu = S/(S+1)$ to balance upload and download bandwidth. We achieve bandwidth $O(n^{1/(S+1)+\epsilon} S \log S)$ by appropriately choosing θ. When the client performs the reconstruction, all irrelevant instances will cancel out, and the answer from the relevant block (i.e., where the queried index belongs) will emerge. We refer to Section 4 for the details of this construction.

Polylogarithmically many servers. The aforementioned S-server scheme has polynomial preprocessing complexity and server space only when S is a constant. This is because we need to choose the field size $q > S$, and the server space is at least q^m. For $m = O(\log n)$, the expression becomes super-polynomial when S is super-constant.

Therefore, to get a scheme with polylogarithmic bandwidth and computation under polylogarithmically many servers, we need a different approach. Specifically, we now encode the database as a polynomial with $m = \mathsf{poly}\log n$ variables with *individual* degree $d = \mathsf{poly}\log n$. Further, we set $S > md$ such that each server only needs to evaluate the original polynomial at one point and need not evaluate any derivatives. To support fast polynomial evaluation in $\mathsf{poly}\log n$ time, we use the polynomial data structure described by Lin, Mook, and Wichs [LMW23] which is in turn a modification of the ideas of Kedlaya and Umans [KU08, KU11]. We describe the full scheme in Appendix B.

3 Preliminaries

We will use the same notations that were set up in Section 2.1. In this section, we will present some additional preliminaries.

3.1 Chain Rule

Given a univariate polynomnial $g \in \mathbb{F}[\lambda]$, we use $g^{(k)}$ to denote the k-th derivative of g, and we use $\overline{\partial}^{(k)} g$ to denote the k-th Hasse derivative of g. We will need to use the chain rule for higher-order derivatives. We first state the chain rule for normal derivatives which was used by Woodruff and Yekhanin's PIR scheme [WY05], and then state the version for Hasse derivates which is the version we will need.

Lemma 3.1 (Chain rule for normal derivatives)**.** *For degree-d m-variate polynomial $f(X_1,\dots,X_m)$ over field $\mathbb{F}$, $\vec{u},\vec{v}\in\mathbb{F}^m$, and let $g(\lambda)=f(\vec{u}+\lambda\vec{v})$ be a univariate polynomial in λ, we have*

$$g^{(k)}(\lambda)=\sum_{l_1,\dots,l_k\in[m]}\frac{\partial^k f}{\partial X_{l_1}\dots\partial X_{l_k}}(\vec{u}+\lambda\vec{v})\prod_{i=1}^{k}v_{l_i}.$$

In the above, the same partial derivative may appear multiple times in the summation depending on the order of the variables; however, in the chain rule for Hasse derivatives, the same partial derivative appears only once as stated below: (for simplicity, we only state the chain rule for multivariate polynomials with individual degree at most 1, which is enough for our scheme)

Lemma 3.2 (Chain rule for Hasse derivatives)**.** *For m-variate polynomial $f(X_1,\dots,X_m)$ with individual degree at most 1 over field F, let $g(\lambda)=f(g_1(\lambda),g_2(\lambda),\dots,g_m(\lambda))$ be a univariate polynomial where each $g_i(\lambda)$ is a polynomial of λ, we have*

$$\overline{\partial}^{(k)}g(\lambda)=\sum_{\vec{a}=(a_1,\dots,a_m)\in A_k}\boldsymbol{\partial}^{\vec{a}}\circ f(g_1(\lambda),g_2(\lambda),\dots,g_m(\lambda))\prod_{i=1}^{m}(g_i^{(1)}(\lambda))^{a_i}.$$

Specifically, for $g(\lambda)=f(\vec{u}+\lambda\vec{v})$, we have

$$\overline{\partial}^{(k)}g(\lambda)=\sum_{\vec{a}=(a_1,\dots,a_m)\in A_k}\boldsymbol{\partial}^{\vec{a}}\circ f(\vec{u}+\lambda\vec{v})\cdot\vec{v}^{\vec{a}}$$

3.2 Prime Field With S-th Root of Unity

In our S-server PIR scheme, we will work with a finite field that has a S-th root of unity. Further, we want the prime field to be small in size for the scheme to be efficient. The following theorem shows that there is a field whose size is $\mathsf{poly}(S)$ with a S-th root of unity; further, all arithmetic operations in the field can be accomplished in $O(\log^2 S)$ time.

Lemma 3.3 (Linnik's Theorem [Lin44, Xyl11a, Xyl11b])**.** *For any integer $S>1$, there exists some prime $q=q(S)$ where $q\equiv 1 \pmod S$ and $q<cS^L$ for some **absolute** constants c,L where $L\le 5$.*

3.3 Polynomial Interpolation and Evaluation

We use $\mathsf{DB}\in\{0,1\}^n$ to denote the database indexed by $0,1,\dots,n-1$. Let $d\le m$ be integers such that $\binom{m}{d}\ge n$. Let $E:\{0,1,\dots,n-1\}\to\{0,1\}^m$ be an an *injective* index function which takes an index $i\in\{0,1,\dots,n-1\}$ and outputs a vector in $\{0,1\}^m$ of Hamming weight exactly d. We can use the following polynomial $F\in\mathbb{F}_q[X_1,\dots,X_m]$ of homogeneous degree d to encode a database $\mathsf{DB}\in\{0,1\}^n$:

$$F(\vec{X})=\sum_{i\in\{0,1,\dots,n-1\}}\mathsf{DB}[i]\cdot\vec{X}^{E(i)}$$

It is easy to see that $F(E(i))=\mathsf{DB}[i]$ for $i\in\{0,1,\dots,n-1\}$.

The following lemma shows that there exists some choice $m=O(\log n)$ and $d=\theta m$ for some constant $\theta\in(0,\frac{1}{2})$, such that $\binom{m}{d}\ge n$:

Lemma 3.4. *For any constant $0<\theta\le 1/2$, if we want $\binom{m}{\theta m}\ge n$ to hold, for sufficiently large n, it suffices to set $m=(1/H(\theta)+o(1))\log(n)$, where $H(p)=-p\log_2 p-(1-p)\log_2(1-p)$ is the binary entropy function.*

The proof of Lemma 3.4 is deferred to Appendix C.

4 Our S-Server PIR Scheme

An S-server PIR scheme $(\mathbf{Preproc}_s, \mathbf{Query}, \mathbf{Answer}, \mathbf{Recons})$ consists of 1) a preprocessing algorithm $\mathbf{Preproc}_s$ indexed by a server index $s \in \{0, 1, \ldots, S-1\}$ which preprocesses the database DB into some encode version, 2) a query algorithm $\mathbf{Query}$ which takes the desired index i and outputs a question for each of the S servers, 3) an answer algorithm denoted $\mathbf{Answer}$ that takes a query and outputs the server's response, and 4) a reconstruction algorithm $\mathbf{Recons}$ that takes in all servers' responses and outputs the answer. We defer the full definition of PIR to Appendix A since the definitions are standard. In this section, we formally describe our S-server scheme.

4.1 Construction

Parameters and notation. We will choose the following parameters.

- Suppose that the n-bit database is partitioned into $B := n^{1-\mu}$ blocks each with n^μ bits. Without loss of generality, we assume that $B := n^{1-\mu}$ is an integer.
- Let $\mathbb{F}_q$ be a field of prime order q which has an S-th root of unity denoted ω. By Lemma 3.3, q is bounded by $\mathsf{poly}(S)$.
- We will encode each block as an m-variate polynomial of homogeneous degree d. We will choose $m = O(\log n)$ and $d = \theta m$ for some constant $0 < \theta \le 1/2$, such that $\binom{m}{d} \ge n^\mu$ — this is possible due to Lemma 3.4. We will also choose d such that it is *not a multiple of* S — jumping ahead, this will be important for the irrelevant blocks to cancel out during reconstruction.
- We use the following polynomial F_j over $\mathbb{F}_q$ to encode each block $j \in \{0, 1, \ldots, B-1\}$:

$$F_j(\vec{X}) = \sum_{i \in \{0,1,\ldots,n^\mu - 1\}} \mathsf{DB}[j \cdot B + i] \cdot \vec{X}^{E(i)}$$

  where $E : \{0, 1, \ldots, n^\mu - 1\} \to \{0,1\}^m$ be an an *injective* index function which takes an index $i \in \{0, 1, \ldots, n^\mu - 1\}$ and outputs a vector in $\{0,1\}^m$ of Hamming weight exactly d. Clearly this map E can be chosen such that $E(i)$ can be evaluated in time $\mathsf{poly}\log n$.

Protocol. Our S-server PIR works as follows.

- $\mathbf{Preproc}_s$: For each block j, each $\vec{a} \in A_{\le \lfloor d/S \rfloor}$, each $\vec{x} \in \mathbb{F}_q^m$, calculate $\omega^{s \cdot \mathsf{wt}(\vec{a})}(\boldsymbol{\partial}^{\vec{a}} \circ F_j(\vec{x}))$, and store all results.
- $\mathbf{Query}$: Let $i \in \{0, 1, \ldots, n-1\}$ be the queried index. Let vector $\vec{u} = E(i \bmod n^\mu) \in \mathbb{F}_q^m$, and let $r = \lfloor i/n^\mu \rfloor$ be the block where i resides. The client randomly picks $\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_{B-1} \in \mathbb{F}_q^m$. For $s \in \{0, 1, \ldots, S-1\}$, the client sets

$$\vec{z}_{j,s} = \begin{cases} \omega^s \vec{v}_j & \text{if } j \ne r \\ \vec{u} + \omega^s \vec{v}_j & \text{if } j = r \end{cases}$$

  The client sends $Q_s = (\vec{z}_{0,s}, \ldots, \vec{z}_{B-1,s})$ to each server $s \in \{0, \ldots, S-1\}$.
- $\mathbf{Answer}$: The s-th server parses the message received from the client as $(\vec{z}_{0,s}, \ldots, \vec{z}_{B-1,s})$. For each $\vec{a} \in A_{\le \lfloor d/S \rfloor}$, it calculates

$$\mathsf{ans}_{s,\vec{a}} = \sum_{j=0}^{B-1} \underbrace{\omega^{s \cdot \mathsf{wt}(\vec{a})}(\boldsymbol{\partial}^{\vec{a}} \circ F_j(\vec{z}_{j,s}))}_{\text{precomputed during preproc}}$$

and sends back $\{\mathsf{ans}_{s,\vec{a}}\}_{\vec{a}\in A_{\lfloor d/S \rfloor}}$ back to the client.

- **Recons**:

  1. Define univariate polynomial $f(\lambda) = F_r(\vec{u} + \lambda\vec{v})$ and $g(\lambda) = \sum_{s=0}^{S-1} f(\omega^s\lambda)$. Given the responses of all servers, the client computes $\overline{\partial}^{(k)} g(1)$ for all $0 \le k \le \lfloor d/S \rfloor$ by:

$$\overline{\partial}^{(k)} g(1) = \sum_{\vec{a}\in A_k} \left(\sum_{s=0}^{S-1} \mathsf{ans}_{s,\vec{a}} \right) \vec{v}_r^{\,\vec{a}}. \tag{1}$$

  2. Reconstruct g by its Hasse derivatives. It is obvious that g has degree at most d. In Lemma 4.4, we will see that g only contains monomials whose degree is a multiple of S. So we can expand $g(\lambda)$ as $\sum_{k=0}^{\lfloor d/S \rfloor} c_k \lambda^{S\cdot k}$, where $\{c_k\}$ are the coefficients to reconstruct. We have

$$\begin{cases} \sum_{k=0}^{\lfloor d/S \rfloor} \binom{S\cdot k}{0} c_k & = \overline{\partial}^{(0)} g(1) \\ \sum_{k=0}^{\lfloor d/S \rfloor} \binom{S\cdot k}{1} c_k & = \overline{\partial}^{(1)} g(1) \\ \cdots \\ \sum_{k=0}^{\lfloor d/S \rfloor} \binom{S\cdot k}{l} c_k & = \overline{\partial}^{(l)} g(1) \\ \cdots \\ \sum_{k=0}^{\lfloor d/S \rfloor} \binom{S\cdot k}{\lfloor d/S \rfloor} c_k & = \overline{\partial}^{(\lfloor d/S \rfloor)} g(1). \end{cases} \tag{2}$$

  The client uses Gaussian elimination to reconstruct the coefficients, and outputs $\frac{1}{S}g(0)$ as the answer.

4.2 Proof of Correctness

It it not hard to see that $\frac{1}{S}g(0) = F_r(E(i \bmod B)) = \mathsf{DB}[i]$, so it suffices to show that **Recons** reconstructs the polynomial g successfully.

First, we show that when the client sums up the corresponding components obtained from the S servers for the same $\boldsymbol{\partial}^{\vec{a}}$ operator, the terms not related to block $r = \lfloor i/n^{\mu} \rfloor$ disappear, as stated in the following lemma.

Lemma 4.1. *For each $\vec{a} \in A_{\le \lfloor d/S \rfloor}$,*

$$\sum_{s=0}^{S-1} \mathsf{ans}_{s,\vec{a}} = \sum_{s=0}^{S-1} \omega^{s\cdot \mathsf{wt}(\vec{a})} (\boldsymbol{\partial}^{\vec{a}} \circ F_r(\vec{u}_r + \omega^s \vec{v}_r)).$$

Notice that on the right-hand side, only terms pertaining to the block r remain.

Proof of Lemma 4.1. To prove Lemma 4.1, it suffices to show the following claim:

Claim 4.2. *For any m-variate polynomial F over $\mathbb{F}_q$ of homogeneous degree d where d* ***is not a multiple of*** *S, any $\vec{v} \in \mathbb{F}_q^m$, and $\vec{a} \in A_{\le d}$,*

$$\sum_{s=0}^{S-1} \omega^{s\cdot \mathsf{wt}(\vec{a})} (\boldsymbol{\partial}^{\vec{a}} \circ F(\omega^s \vec{v})) = 0.$$

To show the above claim, observe that

$$\begin{aligned}\sum_{s=0}^{S-1}\omega^{s\cdot \mathsf{wt}(\vec{a})}(\boldsymbol{\partial}^{\vec{a}}\circ F(\omega^s\vec{v})) &= \sum_{s=0}^{S-1}\omega^{s\cdot \mathsf{wt}(\vec{a})+s\cdot(d-\mathsf{wt}(\vec{a}))}(\boldsymbol{\partial}^{\vec{a}}\circ F(\vec{v}))\\ &= \sum_{s=0}^{S-1}\omega^{s\cdot d}(\boldsymbol{\partial}^{\vec{a}}\circ F(\vec{v}))\\ &= 0,\end{aligned}$$

where the first line is because $\boldsymbol{\partial}^{\vec{a}}\circ F$ has homogeneous degree $d-\mathsf{wt}(\vec{a})$, and the last line is because d is not a multiple of S and recall that ω is the S-th root of unity. □

We now show that Equation (1) correctly reconstructs the Hasse derivatives of g, as stated in the following lemma.

Lemma 4.3. *The Hasse derivatives $\overline{\partial}^{(k)}g(1)$ are reconstructed correctly in Equation* (1).

Proof. By the chain rule of Hasse derivatives (see Lemma 3.2), we have for each $0\le k\le\lfloor d/S\rfloor$,

$$\begin{aligned}\overline{\partial}^{(k)}g(1) &= \sum_{s=0}^{S-1}\sum_{\vec{a}\in A_k}\left(\boldsymbol{\partial}^{\vec{a}}\circ F_r(\vec{u}_r+\omega^s\vec{v}_r)\right)(\omega^s\vec{v}_r)^{\vec{a}}\\ &= \sum_{\vec{a}\in A_k}\left(\sum_{s=0}^{S-1}\omega^{s\cdot k}(\boldsymbol{\partial}^{\vec{a}}\circ F_r(\vec{u}_r+\omega^s\vec{v}_r))\right)\vec{v}_r^{\vec{a}}.\end{aligned}$$

Together with Lemma 4.1, we get

$$\overline{\partial}^{(k)}g(1) = \sum_{\vec{a}\in A_k}\left(\sum_{s=0}^{S-1}\mathsf{ans}_{s,\vec{a}}\right)\vec{v}_r^{\vec{a}}.$$

□

Next we consider the correctness of the second step of **Recons**. First we have the following lemma:

Lemma 4.4. *The degree of each nonzero monomial in g should be a multiple of S.*

Proof. Henceforth, we use $\mathsf{Coeff}_{\lambda^k}(g(\lambda))$ to denote the coefficient of the monomial λ^k in $g(\lambda)$. We thus have the following:

$$\begin{aligned}\mathsf{Coeff}_{\lambda^k}(g(\lambda)) &= \sum_{s=0}^{S-1}\mathsf{Coeff}_{\lambda^k}(f(\omega^s\lambda))\\ &= \sum_{s=0}^{S-1}\omega^{s\cdot k}\mathsf{Coeff}_{\lambda^k}(f(\lambda))\\ &= \mathbf{1}_{S|k}\cdot S\cdot \mathsf{Coeff}_{\lambda^k}(f(\lambda)),\end{aligned}$$

where $\mathbf{1}_{S|k}=1$ if k is divisible by S and 0 otherwise. □

Next, we prove that the linear system shown in Equation (2) has a unique solution, and thus Gaussian elimination works.

Lemma 4.5. *The linear system shown in Equation* (2) *has a unique solution.*

Proof. Define a $(\lfloor d/S \rfloor + 1) \times (\lfloor d/S \rfloor + 1)$ matrix M where $M_{l,k} = \binom{S \cdot k}{l}$ for $0 \le l, k \le \lfloor d/S \rfloor$. Then, we can rewrite the linear system of Equation (2) in a matrix form:

$$M(c_0, \ldots, c_{\lfloor d/S \rfloor})^T = \left(\overline{\partial}^{(0)} g(1), \ldots, \overline{\partial}^{(\lfloor d/S \rfloor)} g(1)\right)^T.$$

It suffices to show that M is invertible over $\mathbb{F}_q$, as stated in the following claim.

Claim 4.6. *For any $(h+1) \times (h+1)$ matrix M where $M_{l,k} = \binom{S \cdot k}{l}$ for $0 \le l, k \le h$, we have $\det(M) = S^{h(h+1)/2}$ which is nonzero in any prime field $\mathbb{F}_q$ where q is co-prime to S.*

Proof. For $0 \le k \le h$, we will use the notation $M[:k]$ to denote the k-th column vector of matrix M.

Let M' be a $(h+1) \times (h+1)$ matrix such that: $M'[:k] = \sum_{t=0}^{k} (-1)^{k-t} \binom{k}{t} M[:t]$ for $0 \le k \le h$. We have:

$$\begin{aligned}
M'_{l,k} &= \sum_{t=0}^{k} (-1)^{k-t} \binom{k}{t} M_{l,t} \\
&= \sum_{t=0}^{k} (-1)^{k-t} \binom{k}{t} \binom{S \cdot t}{l} \\
&= \sum_{t=0}^{k} (-1)^{k-t} \binom{k}{t} \mathsf{Coeff}_{x^l}\left((x+1)^{S \cdot t}\right) \\
&= \mathsf{Coeff}_{x^l}\left(\sum_{t=0}^{k} (-1)^{k-t} \binom{k}{t} (x+1)^{S \cdot t}\right) \\
&= \mathsf{Coeff}_{x^l}\left(\left((x+1)^S - 1\right)^k\right) \\
&= \mathsf{Coeff}_{x^l}\left(\left(\sum_{t=0}^{S} \binom{S}{t} x^t - 1\right)^k\right) \\
&= \mathsf{Coeff}_{x^l}\left(\left(\sum_{t=1}^{S} \binom{S}{t} x^t\right)^k\right) \\
&= \mathsf{Coeff}_{x^l}\left(x^k f_S(x)^k\right)
\end{aligned}$$

where $f_S(x) = \sum_{t=1}^{S} \binom{S}{t} x^{t-1}$ is a degree-$(S-1)$ polynomial whose constant term is S.

We can see that M' is a lower triangular matrix and $M'_{k,k} = S^k$. Moreover, M' can be obtained by M through a series of elementary column operations: "adding a multiple of one column to another column", so

$$\det(M) = \det(M') = \prod_{k=0}^{h} S^k = S^{h(h+1)/2}.$$

□

□

4.3 Proof of Security

The privacy proof is easy to see: each server $s \in \{0, 1, \ldots, S-1\}$ receives $\vec{z}_{j,s} = \vec{u}_j + \omega^s \vec{v}_j$ for $\vec{v}_j \overset{\$}{\leftarrow} \mathbb{F}_q^m$ for block j, where $\vec{u}_j = \vec{u}$ for $j = r$ and $\vec{u}_j = 0$ for other blocks. Since $\omega^s \not\equiv 0$ in $\mathbb{F}_q$ and each $\vec{v}_j$ is i.i.d. sampled, $\vec{z}_{j,s}$ is also randomly distributed in $\mathbb{F}_q^m$.

4.4 Efficiency

Let $\Lambda(m, w) := \sum_{h=0}^{w} \binom{m}{h}$. We denote $\log$ the logarithmic function with base 2. For $\theta \in [0, 1]$, we denote the *binary entropy* of θ by $H(\theta)$, where $H(\theta) = -\theta \log \theta - (1-\theta) \log(1-\theta)$ for $\theta \in (0, 1)$, and $H(0) = H(1) = 0$.

- **Bandwidth:** For each server s and each block $j \in \{0, 1, \ldots, B-1\}$, the client will send a vector $\vec{z}_{j,s} \in \mathbb{F}_q^m$ to the server. Recall that $m = \mu(1/H(\theta) + o(1)) \log n$ (Lemma 3.4) and each element in $\mathbb{F}_q$ takes $O(\log S)$ space (Lemma 3.3), so the per-server upload bandwidth is

$$O(Bm \log S) = O(n^{1-\mu+o(1)} \log S).$$

  For each server s and each $\vec{a} \in A_{\leq \lfloor d/S \rfloor}$, the server returns an answer $\mathsf{ans}_{s,\vec{a}} \in \mathbb{F}_q$. By the fact that $|A_{\leq \lfloor d/S \rfloor}| = \Lambda(m, \lfloor \theta m/S \rfloor) \leq 2^{H(\theta/S)m}$ for $0 < \theta \leq 1/2$, the per-server download bandwidth is

$$O(|A_{\leq \lfloor d/S \rfloor}| \log S) = O(n^{\mu(H(\theta/S)/H(\theta)+o(1))} \log S).$$

  By the fact that $\frac{H(\theta/S)}{H(\theta)} \to 1/S$ (and $\frac{H(\theta/S)}{H(\theta)} > 1/S$) when $\theta \to 0$, if we choose θ to be sufficiently small, we can achieve $O(n^{\mu/S+\epsilon} \log S)$ download bandwidth for any constant ϵ. To balance the upload and download bandwidth, we may solve the equation $1 - \mu = \mu/S$ and let $\mu = S/(S+1)$. The total per-server bandwidth will become $O(n^{(SH(\theta/S))/((S+1)H(\theta))+o(1)} \log S)$.

  Moreover, $\frac{H(\theta/S)}{H(\theta)} - 1/S = O(\frac{1}{\log \frac{1}{\theta}})$ when $\theta \to 0$, so if we want to achieve $O(n^{1/(S+1)+\epsilon} \log S)$ per-server bandwidth for some $\epsilon > 0$, if suffices to set $\frac{1}{\log \frac{1}{\theta}} = O(\epsilon)$ i.e. $\theta = 1/\exp(O(1/\epsilon))$, and m will become

$$m = \mu(1/H(\theta) + o(1)) \log n = O(1/(-\theta \log \theta)) \log n = \exp(O(1/\epsilon)) \log n.$$

- **Server computation:** For each server s and each $\vec{a} \in A_{\lfloor d/S \rfloor}$, the server needs to add up B elements in $\mathbb{F}_q$ to compute $\mathsf{ans}_{s,\vec{a}}$. By Lemma 3.3, each add operation in $\mathbb{F}_q$ takes time $O(\log S)$, so now the computation of each server is bounded by

$$O(B|A_{\leq \lfloor d/S \rfloor}| \log S) = O(n^{1-\mu+\mu(H(\theta/S)/H(\theta)+o(1))} \log S).$$

  Specifically, if we set $\mu = S/(S+1)$ and $\theta = 1/\exp(O(1/\epsilon))$, the server computation will become

$$O(n^{2/(S+1)+\epsilon} \log S).$$

- **Client computation:** First the client computation is not less than the bandwidth i.e. $O(BmS \log S + |A_{\leq \lfloor d/S \rfloor}| S \log S)$. Then we consider the time complexity of **Recons**.

  Since the Gaussian elimination takes only $\mathsf{poly}(m, \log S) = \mathsf{poly}(\log n, \log S)$ time, the time complexity of **Recons** is bounded by the arithmetic operations in $\mathbb{F}_q$ to reconstruct the Hasse derivatives of g (see Equation (1)).

For each $\vec{a} \in A_{\leq \lfloor d/S \rfloor}$, the client should do $O(m) = n^{o(1)}$ multiplications in $\mathbb{F}_q$ (where each takes $O(\log^2 S)$ time) and $O(S)$ additions in $\mathbb{F}_q$ (where each takes $O(\log S)$ time), the client computation is

$$\begin{aligned} &O(BmS\log S + |A_{\leq \lfloor d/S \rfloor}|(S\log S + m\log^2 S)) \\ =&O((n^{1-\mu+o(1)} + n^{\mu(H(\theta/S)/H(\theta)+o(1))})S\log S). \end{aligned}$$

Specifically, if we set $\mu = S/(S+1)$ and $\theta = 1/\exp(O(1/\epsilon))$, then the client computation is

$$O(n^{1/(S+1)+\epsilon} S\log S).$$

- **Server space:** Recall that we use a precompute-all approach: for each block j, each $\vec{a} \in A_{\leq \lfloor d/S \rfloor}$ and each $\vec{x} \in \mathbb{F}_q^m$, each server stores an element in $\mathbb{F}_q$. So the server space is bounded by

$$O(B|A_{\leq \lfloor d/S \rfloor}|q^m \log S) = O(q^m n^{1-\mu+\mu(H(\theta/S)/H(\theta)+o(1))}\log S) = S^{O(m)}.$$

  Specifically, if we set $\mu = S/(S+1)$ and $\theta = 1/\exp(O(1/\epsilon))$, the server space will become

$$S^{\exp(O(1/\epsilon))\log n}.$$

- **Preprocessing time:** It is not hard to see that each element stored by each server can be computed in $\mathsf{poly}(n, \log S)$ time, so the preprocessing time is still $S^{O(m)}$.

  Specifically, if we set $\mu = S/(S+1)$ and $\theta = 1/\exp(O(1/\epsilon))$, the preprocessing time will become

$$S^{\exp(O(1/\epsilon))\log n}.$$

In conclusion, if we set $\mu = S/(S+1)$ and θ be a sufficiently small constant, we can achieve $O(n^{1/(S+1)+\epsilon}\log S)$ per-server bandwidth, $O(n^{2/(S+1)+\epsilon}\log S)$ per-server computation and $O(n^{1/(S+1)+\epsilon}S\log S)$ client computation for any $\epsilon > 0$, with $S^{\exp(O(1/\epsilon))\log n}$ preprocessing time and server storage. Specifically, for 2-server scheme we can achieve $O(n^{1/3+\epsilon})$ bandwidth and $O(n^{2/3+\epsilon})$ server computation for any $\epsilon > 0$.

Besides, we may choose different μ to balance the bandwidth and server computation. In the above analysis, we can see that the construction in Section 4.1 has

$$O((n^{1-\mu+o(1)} + n^{\mu(H(\theta/S)/H(\theta)+o(1))})\log S)$$

per-server bandwidth,

$$O(n^{1-\mu+\mu(H(\theta/S)/H(\theta)+o(1))}\log S)$$

per-server computation, and

$$O((n^{1-\mu+o(1)} + n^{\mu(H(\theta/S)/H(\theta)+o(1))})S\log S)$$

client computation.

For any $1/(S+1) \leq \alpha \leq 1/S$, if we choose $\mu = S\cdot\alpha$, by the fact that $1-\mu \leq \mu/S$ when $1/(S+1) \leq \alpha$, the scheme has

$$O(n^{S\alpha(H(\theta/S)/H(\theta)+o(1))}\log S)$$

per-server bandwidth,

$$O(n^{1-S\alpha+S\alpha(H(\theta/S)/H(\theta)+o(1))}\log S)$$

per-server computation, and

$$O(n^{S\alpha(H(\theta/S)/H(\theta)+o(1))}S\log S)$$

client computation.

Moreover, similar as the analysis before, it suffices to choose $\theta = 1/\exp(O(1/\epsilon))$ for $H(\theta/S)/H(\theta) < 1/S + \epsilon$ to hold, so we have the following theorem:

Theorem 4.7. *There exists an S-server PIR scheme such that, for any $\epsilon > 0$ and $1/(S+1) \leq \alpha \leq 1/S$, it can achieve $O(n^{\alpha+\epsilon} \log S)$ per-server bandwidth, $O(n^{1-(S-1)\alpha+\epsilon} \log S)$ per-server computation and $O(n^{\alpha+\epsilon} S \log S)$ client computation per query, with $S^{\exp(O(1/\epsilon)) \log n}$ preprocessing time and server storage. Specifically, when S is a constant, the preprocessing time and server storage are bounded by* $\mathsf{poly}(n)$.

References

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *S&P*, 2018.

[BF90] Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *Symposium on Theoretical Aspects of Computer Science*, 1990.

[BFG03] Richard Beigel, Lance Fortnow, and William I. Gasarch. A nearly tight bound for private information retrieval protocols. *Electronic Colloquium on Computational Complexity (ECCC)*, 2003.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In *CRYPTO*, pages 55–73, 2000.

[BS80] Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

[CG97] Benny Chor and Niv Gilboa. Computationally private information retrieval. In *STOC*, 1997.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, 1995.

[Cha04] Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In *ACISP*, 2004.

[CHK22] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In *Eurocrypt*, 2022.

[CK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *EUROCRYPT*, 2020.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, pages 402–414, 1999.

[DG16] Zeev Dvir and Sivakanth Gopi. 2-server pir with subpolynomial communication. *J. ACM*, 63(4), 2016.

[DMO00] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *EUROCRYPT*, 2000.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: scaling private contact discovery. *Proc. Priv. Enhancing Technol.*, 2018(4):159–178, 2018.

[Fea] Nick Feamster. Oblivious DNS deployed by Cloudflare and Apple. https://medium.com/noise-lab/oblivious-dns-deployed-by-cloudflare-and-apple-1522ccf53cab.

[Gas04] William I. Gasarch. A survey on private information retrieval. *Bulletin of the EATCS*, 82:72–107, 2004.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *ICALP*, 2005.

[GZS24] Ashrujit Ghoshal, Mingxun Zhou, and Elaine Shi. Efficient pre-processing pir without public-key cryptography. In *Eurocrypt*, 2024.

[hav] https://haveibeenpwned.com/.

[HDCG$^+$23] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, , and Nickolai Zeldovich. Private web search with Tiptoe. In *29th ACM Symposium on Operating Systems Principles (SOSP)*, Koblenz, Germany, October 2023.

[HHCG$^+$23] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *Usenix Security*, 2023.

[HPPY24] Alexander Hoover, Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Plinko: Single-server PIR with efficient updates via invertible prfs. *IACR Cryptol. ePrint Arch.*, page 318, 2024.

[KCG21] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In *Usenix Security*, 2021.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *FOCS*, 1997.

[KU08] Kiran S. Kedlaya and Christopher Umans. Fast modular composition in any characteristic. In *49th Annual IEEE Symposium on Foundations of Computer Science*, pages 146–155, 2008.

[KU11] Kiran S. Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 2011.

[Lin44] UV Linnik. On the least prime in an arithmetic progression. i. the basic theorem. *Matematicheskiy sbornik*, 15(2):139–178, 1944.

[Lip09] Helger Lipmaa. First CPIR protocol with data-dependent computation. In *ICISC*, 2009.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 595–608. ACM, 2023.

[LP22] Arthur Lazzaretti and Charalampos Papamanthou. Single server pir with sublinear amortized time and polylogarithmic bandwidth. Cryptology ePrint Archive, Paper 2022/830, 2022. https://eprint.iacr.org/2022/830.

[LP23] Arthur Lazzaretti and Charalampos Papamanthou. Treepir: Sublinear-time and polylog-bandwidth private information retrieval from ddh. In *CRYPTO*, 2023.

[MCG$^+$08] Carlos Aguilar Melchor, Benoit Crespin, Philippe Gaborit, Vincent Jolivet, and Pierre Rousseau. High-speed private information retrieval computation on GPU. In *Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*, SECURWARE '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[MCR21] Muhammad Haris Mughees, Hao Chen, and Ling Ren. Onionpir: Response efficient single-server pir. In *CCS*. Association for Computing Machinery, 2021.

[MG07] Carlos Aguilar Melchor and Philippe Gaborit. A lattice-based computationally-efficient private information retrieval protocol. *IACR Cryptology ePrint Archive*, 2007:446, 2007.

[MIR23] Muhammad Haris Mughees, Sun I, and Ling Ren. Simple and practical amortized sublinear private information retrieval. Cryptology ePrint Archive, Paper 2023/1072, 2023.

[MW22] Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *IEEE S&P*, 2022.

[obl] Oblivious dns over https. https://tools.ietf.org/html/draft-pauly-dprive-oblivious-doh-04.

[OG11] Femi G. Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Financial Cryptography*, pages 158–172, 2011.

[OS07] Rafail Ostrovsky and William E. Skeith, III. A survey of single-database private information retrieval: techniques and applications. In *PKC*, pages 393–411, 2007.

[RY06] Alexander A. Razborov and Sergey Yekhanin. An $\omega(n^{1/3})$ lower bound for bilinear group based private information retrieval. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 739–748, 2006.

[SACM21] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce Maggs. Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In *CRYPTO*, 2021.

[SC07] Radu Sion and Bogdan Carbunar. On the computational practicality of private information retrieval. In *Network and Distributed Systems Security Symposium (NDSS)*, 2007.

[SCV$^+$21] Sudheesh Singanamalla, Suphanat Chunhapanya, Marek Vavruša, Tanya Verma, Peter Wu, Marwan Fayed, Kurtis Heimerl, Nick Sullivan, and Christopher Wood. Oblivious dns over https (odoh): A practical privacy enhancement to dns. In *PET Symposium*, 2021.

[sig] Technology deep dive: Building a faster oram layer for enclaves. https://signal.org/blog/building-faster-oram/.

[WY05] David P. Woodruff and Sergey Yekhanin. A geometric approach to information-theoretic private information retrieval. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 275–284. IEEE Computer Society, 2005.

[Xyl11a] Triantafyllos Xylouris. On the least prime in an arithmetic progression and estimates for the zeros of dirichlet l-functions. *Acta Arithmetica*, 150(1):65–91, 2011.

[Xyl11b] Triantafyllos Xylouris. *Über die Nullstellen der Dirichletschen L-Funktionen und die kleinste Primzahl in einer arithmetischen Progression*. PhD thesis, Universitäts-und Landesbibliothek Bonn, 2011.

[ZLTS23] Mingxun Zhou, Wei-Kai Lin, Yiannis Tselekounis, and Elaine Shi. Optimal single-server private information retrieval. In *EUROCRYPT*, 2023.

[ZPSZ24] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server pir with sublinear server computation. In *IEEE S& P*, 2024.

A Definitions: S-Server PIR with Global Preprocessing

We give a formal definition of an S-server information-theoretic PIR with global preprocessing. We index the servers by $0, 1, \ldots, S-1$.

Definition A.1 (S-server PIR)**.** An S-server PIR scheme consists of the following possibly randomized algorithms:

- $\widetilde{\mathsf{DB}}_s \leftarrow \mathbf{Preproc}_s(\mathsf{DB})$: given database $\mathsf{DB} \in \{0,1\}^n$, server $s \in \{0, 1, \ldots, S-1\}$ calls this algorithm to do a one-time preprocessing and computes an encoding of the database denoted $\widetilde{\mathsf{DB}}_s$.
- $st, Q_0, \ldots, Q_{S-1} \leftarrow \mathbf{Query}(n, i)$: given the database size n and a query index $i \in \{0, 1, \ldots, n-1\}$, the algorithm outputs some private state st as well as $Q_0, \ldots, Q_{S-1}$ representing the query messages to be sent to each of the S servers.
- $\mathbf{Answer}(\widetilde{\mathsf{DB}}_s, Q_s)$: given the encoded database $\widetilde{\mathsf{DB}}_s$ and a query message Q_s, this algorithm outputs the response message ans_s;
- $\mathbf{Recons}(st, \mathsf{ans}_0, \ldots, \mathsf{ans}_{S-1})$: given the private state st and the responses $\mathsf{ans}_0, \ldots, \mathsf{ans}_{S-1}$ from all the servers, this algorithm reconstructs the answer $\mathsf{DB}[i]$.

The scheme should satisfy the following properties:

Correctness. Correctness requires that the client should output the correct answer under an honest execution. Formally, we want that for any n, $\mathbf{DB} \in \{0,1\}^n$ and $i \in \{0, 1, \ldots, n-1\}$,

$$\Pr\left[\begin{array}{ll} \forall s \in \{0, \ldots, S-1\} : \widetilde{\mathsf{DB}}_s \leftarrow \mathbf{Preproc}_s(\mathsf{DB}), & \\ st, Q_0, \ldots, Q_{S-1} \leftarrow \mathbf{Query}(n, i), & : \mathbf{Recons}(st, \mathsf{ans}_0, \ldots, \mathsf{ans}_{S-1}) = \mathsf{DB}[i] \\ \forall s \in \{0, \ldots, S-1\} : \mathsf{ans}_s \leftarrow \mathbf{Answer}(\widetilde{\mathsf{DB}}_s, Q_s) & \end{array}\right] = 1$$

Security. Security requires that any individual server's view leaks nothing about the client's desired index. Formally, for any n, S, for any $i_1, i_2 \in \{0, 1, \ldots, n-1\}$ and any $s \in \{0, \ldots, S-1\}$, the distributions $\{Q_s : (Q_0, \ldots, Q_{S-1}) \leftarrow \mathbf{Query}(n, i_1)\}$ and $\{Q_s : (Q_0, \ldots, Q_{S-1}) \leftarrow \mathbf{Query}(n, i_2)\}$, are identical.

B Achieving Polylogarithmic Bandwidth and Computation with Polylogarithmically Many Servers

In this section, we describe a PIR scheme that achieves polylogarithmic bandwidth and computation with polylogarithmically many servers.

Following our approach in the previous sections, if we set $S > d = \Omega(\log n)$, each server only has to return one element to the client, i.e., we achieve $\mathsf{poly}\log n$ bandwidth. However, the server storage would become $S^{O(\log n)}$ which is super-polynomial in n.

To get a PIR scheme with polylogarithmic bandwidth and computation under polylogarithmically many servers, we need to use a different data structure that support fast polynomial evaluation. Specifically, we use the data structure presented by Lin, Mook, and Wichs [LMW23] which is a modification of the ideas by Kedlaya and Umans [KU08, KU11].

B.1 Additional Preliminaries: Data Structure for Fast Polynomial Evaluation

In Lin et al. [LMW23, Appendix A], the authors describe the following data structure. Let F be an m-variate polynomial of individual degree less than d over prime field $\mathbb{F}_q$, then they show that:

- There is a preprocessing algorithm that takes coefficients of F as input and runs in time $d^m \cdot O((m(\log m + \log d + \log\log q))^m) \cdot \mathsf{poly}(m, d, \log q)$ and outputs a data structure of size at most $d^m \cdot O((m(\log m + \log d + \log\log q))^m) \cdot \mathsf{poly}(m, d, \log q)$,
- Moreover, there is an evaluation algorithm with random access to the data structure that evaluates F at any point with $O(\mathsf{poly}(d, m, \log q))$ time.

The authors also present an interpolation algorithm [LMW23, Lemma 2.2], in order to encode a database into such polynomial F:

- Given $d \le q$, there is an interpolation algorithm that takes d^m values $\{y_{(x_1,\ldots,x_m)}\}_{(x_1,\ldots,x_m)\in\{0,1,\ldots,d-1\}^m}$, and recovers coefficients of a polynomial $F(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$ with individual degree $< d$ in each variable such that $F(x_1, \ldots, x_m) = y_{(x_1,\ldots,x_m)}$ for all $(x_1, \ldots, x_m) \in \{0, 1, \ldots, d-1\}^m$. Further, the algorithm runs in time $O(d^m \cdot m \cdot \mathsf{poly}\log q)$.

Remark B.1. The original version of the algorithm in Lin et al. [LMW23] only supports the case $d = q$, but it can be trivially extended to $d \le q$, since fast univariate polynomial interpolation algorithm can be implemented in any field $\mathbb{F}_q$.

B.2 Construction

We now present a PIR scheme using polylogarithmically many servers. Specifically, we will set $S > md$ such that each server only evaluates the original polynomial at one point and need not evaluate any derivatives.

Parameters and notation. Let $\epsilon > 0$ be a constant, set $m = \lceil \log n/(\epsilon \log\log n)\rceil$, $d = \lceil n^{1/m}\rceil = \lceil \log^\epsilon n\rceil$ such that $d^m \ge n$, and $S = md + 1 = O(\log^{1+\epsilon} n/\log\log n)$. We pick the smallest prime q such that $q > S$ and work in field $\mathbb{F}_q$. By Bertrand's postulate, it holds that $q = O(S)$.

S-server PIR. Our S-server PIR works as follows.

- $\mathbf{Preproc}_s$: Encode database DB to m-variate polynomial F with individual degree $< d$. Concretely, we construct $E : \{0, 1, \ldots, n-1\} \to \{0, 1, \ldots, d-1\}^m$ be an *injective* index function, and recover F by interpolating on the set $\{y_{E(i)}\}_{i\in\{0,1,\ldots,n-1\}}$ using the techniques described by Lin et al. [LMW23]. Clearly E exists since $d^m \ge n$. Then, use the preprocessing algorithm from [LMW23, Appendix A] to create a data structure that supports fast evaluation of F at any point.

- **Query**: Let $i \in \{0, 1, \ldots, n-1\}$ be the queried index. Let vector $\vec{u} = E(i)$, the client randomly picks $\vec{v} \in \mathbb{F}_q^m$. For each $s \in \{0, 1, \ldots, S-1\}$, the client sets

$$\vec{z}_s = \vec{u} + \lambda_s \vec{v}.$$

  where $\lambda_s = s + 1$ (we can choose $\{\lambda_0, \ldots, \lambda_{S-1}\}$ to be any S distinct and nonzero points). The client sends $Q_s = \vec{z}_s$ to each server $s \in \{0, \ldots, S-1\}$.

- **Answer**$_s$: The s-th server parses the message received from the client as a vector $\vec{z}_s$. Then calculates

$$\mathsf{ans}_s = F(\vec{z}_s)$$

  and sends back ans_s to the client.

- **Recons**: Given the responses of all servers, the client computes and outputs

$$\sum_{s=0}^{S-1} \mathsf{ans}_s l_s(0) \tag{3}$$

  where $l_s(\lambda) = \prod_{0 \le j < S, j \ne s} \frac{\lambda - \lambda_j}{\lambda_s - \lambda_j}$ is the s-th Lagrange basis polynomial.

B.3 Proof of Correctness

Define univariate polynomial $f(\lambda) = F(\vec{u} + \lambda \vec{v})$, then $f(\lambda_s) = \mathsf{ans}_s = F(\vec{z}_s)$. The degree of f is at most $md < S$ and $f(0) = F(\vec{u}) = F(E(i)) = \mathsf{DB}[i]$.

Hence, by the uniqueness of univariate polynomial interpolation, f is fully determined by its values on any distinct S points. Specifically, given $\{f(\lambda_s)\}$, one can interpolate f by the standard Lagrange interpolation algorithm:

$$f(\lambda) = \sum_{s=0}^{S-1} f(\lambda_s) l_s(\lambda)$$

Substituting λ by 0, we obtain Equation (3).

B.4 Proof of Security

Each server $s \in \{0, 1, \ldots, S-1\}$ receives $\vec{z}_s = \vec{u} + \lambda_s \vec{v}$, the privacy follows the fact $\vec{z}_s$ is randomly distributed in $\mathbb{F}_q^m$ when $\lambda_s \ne 0$ and $\vec{v}$ is randomly sampled.

B.5 Efficiency

We now analyze the efficiency of our construction.

- **Bandwidth:** Each server s receives a vector $\vec{z}_s \in \mathbb{F}_q^m$ and sends back $\mathsf{ans}_s \in \mathbb{F}_q$, so the per-server bandwidth is bounded by $O(m \log q) = O(\log n)$.

- **Server computation:** Each server s computes $F(\vec{z}_s)$. From the result of [LMW23, Appendix A], this makes server computation $O(\mathsf{poly}(d, m, \log q)) = O(\mathsf{poly} \log n)$.

- **Client computation:** Bandwidth is part of client computation, which is $O(Sm \log q)$. The client needs to compute the $l_s(0)$ for $s \in \{0, 1, \ldots, S\}$. Computing each of involves a product of S terms and an inverse of such product, i.e., computing each $l_s(0)$ takes $O(S \log^2 q)$ operations, and computing all of them therefore takes $O(S^2 \log^2 q)$ operations. Finally, the client needs to compute the sum $\sum_{s=0}^{S-1} \mathsf{ans}_s l_s(0)$. This takes $O(S \log^2 q)$ operations. Therefore, the total client computation is $O(SmT \log q + S^2 \log^2 q) = O(\log^{2+2\epsilon} n)$.

- **Server space:** The server needs to store the data structure for F that allows for fast evaluation of F. It follows from [LMW23, Appendix A], that this takes space $d^m \cdot O((m(\log m + \log d + \log \log q))^m) \cdot \mathsf{poly}(m, d, \log q) = n^{(1+1/\epsilon)(1+o(1))}$.

- **Preprocessing time:** Each server should first interpolate the polynomial F and compute the data structure. By [LMW23, Appendix A], the preprocessing time is

$$\begin{aligned} &O(d^m \cdot m \cdot \mathsf{poly} \log q) + d^m \cdot O((m(\log m + \log d + \log \log q))^m) \cdot \mathsf{poly}(m, d, \log q) \\ =&n^{(1+1/\epsilon)(1+o(1))}. \end{aligned}$$

In conclusion, we have:

Theorem B.2. *There exists an $O(\log^{1+\epsilon} n / \log \log n)$-server PIR scheme for any $\epsilon > 0$ such that, it can achieve $O(\log n)$ per-server communication, $O(\mathsf{poly} \log n)$ per-server computation and $O(\log^{2+2\epsilon} n)$ client computation per query, with $n^{(1+1/\epsilon)(1+o(1))}$ preprocessing time and server storage.*

C Proof of Lemma 3.4

We now prove Lemma 3.4. By Stirling's approximation, we have

$$\binom{m}{\theta m} \geq \frac{2^{H(\theta)m}}{\sqrt{2\pi m \theta (1-\theta)}} - o(1) = 2^{H(\theta)m - O(\log m)} = 2^{(H(\theta) - o(1))m}.$$

So it suffices to set $m = (1/H(\theta) + o(1)) \log n$.

D Additional Related Work

So far, we reviewed related work on information theoretic PIR in the global preprocessing model. We now review additional related work including computationally secure schemes and PIR schemes in the client-specific preprocessing model.

Computationally secure PIR schemes. In this paper, we focus on information-theoretic PIR. In either the classical setting or the global preprocessing setting, to achieve information theoretic security, we need at least two servers due to well-known lower bounds [DMO00]. It is known, however, that with suitable computational assumptions, we can get a single-server PIR scheme with polylogarithmic bandwidth and computation per query, assuming polynomial amount of server space [LMW23]. Further, in the classical setting, various works showed how to construct a computationally secure single-server PIR scheme with sublinear bandwidth [CG97, CMS99, KO97, HHCG+23, MW22]. There have also been various attempts at implementing these schemes and making them practical [HHCG+23, MW22, ACLS18, HDCG+23, MCR21].

The client-specific preprocessing model. Although our work focuses on the global preprocessing model, it is worth noting that a flurry of recent results have showed more efficient constructions in the client-specific model [CK20,CHK22,ZLTS23,LP23,LP22,ZPSZ24,GZS24,KCG21,HPPY24,MIR23], including efficient implementations [ZPSZ24, GZS24, LP23, KCG21, MIR23]. As mentioned, in comparison, the global preprocessing model enjoys some advantages such as the ability to amortize the preprocessing overhead among many clients, and better practicality for fast evolving databases.