Scalable Multi-Server Private Information Retrieval*

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 revisit multi-server Private Information Retrieval (PIR), where the client interacts with S non-colluding servers. Ideally, we want a *scalable* family of multi-server PIR schemes where all the performance metrics of the scheme decrease as S increases. However, no prior work achieved scalability under any setting, and any hardness assumption.

In this paper we construct new multi-server, information-theoretically secure *scalable* PIR schemes for three natural settings. First, we give a construction where all the performance metrics scale at equal rate. Second, we give a scalable construction that minimizes the per-query bandwidth. Third, we give a scalable construction that minimizes the per-query online bottleneck cost (the maximum of the bandwidth and computation). For the first two settings, our constructions are *doubly efficient* with only a super-constant number of servers. In comparison, the best known prior works in the information-theoretic setting required super-logarithmically many servers to achieve the doubly efficient notion.

Our techniques for achieving scalable PIR also enable us to advance the state of the art in the polynomial space setting. In this setting, we show how to improve the space consumption of prior works by a polynomial factor while preserving all other metrics. Further, we show a new balancing technique that allows us to further minimize the bandwidth per query by trading off the computation and server space, thus enabling a more smooth tradeoff between the metrics and generalizing the design space.

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 more.

In this paper, we revisit multi-server PIR, where a client interacts with S non-colluding servers. This setting becomes particularly interesting if adding more servers can improve the performance of PIR schemes.

*An online version is available at https://eprint.iacr.org/2024/765.

†Author ordering is randomized.

The holy grail is to get a *scalable* family of schemes, where the all performance metrics, including computation, bandwidth, and space decrease as we increase the number of servers.

Unfortunately, to the best of our knowledge, *no prior multi-server PIR scheme can achieve scalability under any setting and any hardness assumptions*. Classical PIR schemes without preprocessing [CGKS95, Cha04, GR05, CMS99, CG97, KO97, Lip09, OS07, Gas04, BFG03, SC07, OG11, MCG+08, MG07, HHCG+23, MW22] inherently cannot scale beyond the linear computation barrier. Specifically, Beimel, Ishai, and Malkin [BIM04] proved that any classical PIR scheme (without preprocessing) must incur a server-side computation cost that is linear in the size of the database for every query. Intuitively, if the servers need not look at some locations to answer a client's query, then the client's interest in those locations can be ruled out, thus breaking privacy. Although recent works, pioneered by Beimel et al. [BIM04] and Corrigan-Gibbs and Kogan [CK20], have shown how to overcome this linear computation barrier by introducing a *one-time preprocessing* [BIM04, CK20, WY05, LP23, SACM21, CHK22, ZLTS23, LP22, GZS24, HPPY24, MIR23, LMW23], existing preprocessing schemes fail to satisfy scalability. Most existing preprocessing PIR schemes [CK20, LP23, SACM21, CHK22, ZLTS23, LP22, GZS24, HPPY24, MIR23, LMW23] are designed for a fixed number of servers (e.g., one or two), making scalability beyond their scope. While a few works [BIM04, WY05, SWZ24] have explored a parameterizable number of servers, none have achieved full scalability. For example, Beimel et al. [BIM04] and Woodruff and Yekhanin [WY05] showed that bandwidth and computation can scale as fast as $n^{O(1/S)}$ where S is the number of servers. However, their server space grows as rapidly as $n^{\omega(S^2/\log S)}$, which is the opposite of scaling. The concurrent and independent work of Singh et al. [SWZ24] showed how to achieve $n^{O(1/S)}$ bandwidth scaling, but their computation and client space cannot scale beyond $n^{1/2}$.

In this paper, we ask the following natural question:

Can we have multi-server PIR schemes with scalable efficiency as we increase the number of servers?

1.1 Our Main Results

We answer the above question affirmatively. Since tradeoffs exist between the bandwidth, computation, and space requirement of the PIR scheme, we present our results for three natural settings:

- *Setting 1: equal scaling.* What scalability can we achieve if we want all metrics to scale equally fast w.r.t. the number of servers? The concept of equal scaling is similar to that of "bottleneck cost" proposed in the MPC literature [BJPY18]. The idea is that as the problem size scales up, the most unscalable dimension becomes the bottleneck. This setting is particularly of relevance when the storage, computation and network transmission costs of a system are similar.
- *Setting 2: minimize bandwidth.* How do we minimize the bandwidth consumption per query while achieving scalability in all dimensions? This setting is particularly of relevance when the network transmission costs of a system overwhelmingly dominate compute and storage costs. This was the primary setting considered in prior works such as Beimel et al. [BIM04] and Woodruff and Yekhanin [WY05]; thus results in this setting allows a direct comparison with these prior works.
- *Setting 3: minimize bottleneck cost.* How do we minimize the bottleneck cost per query (i.e., the max of the bandwidth and computation) while achieving scalability in server space? This setting is particularly of relevance when storage is relatively cheap and we want to optimize the query response times, subject to being scalable.

Below, we state our main results for these three settings. Unless otherwise stated, the notation $\widetilde{O}(\cdot)$ hides polylogarithmic dependences on n and S. We also omit $O(\log S/\log n)$ terms in the exponent of n since typically $S = n^{o(1)}$.

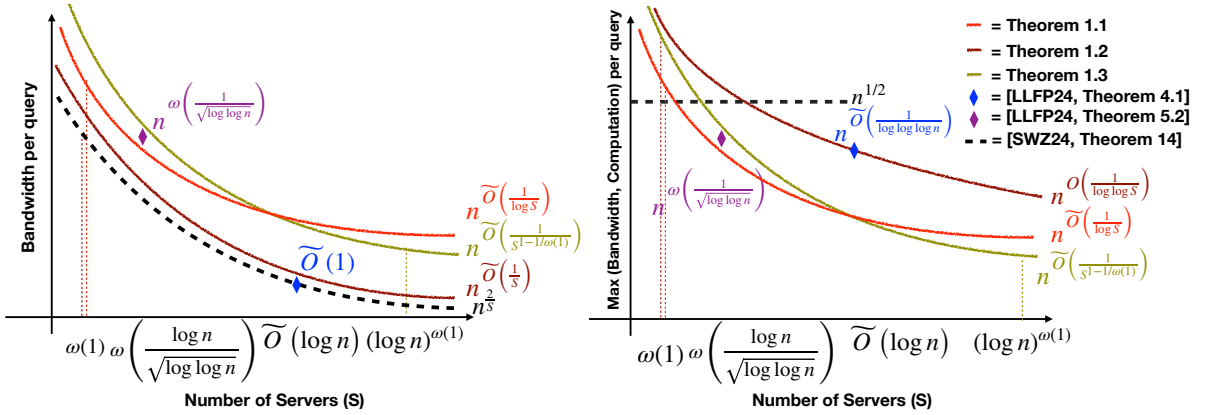

Figure 1: **Left**: Graph comparing the bandwidth of our doubly efficient constructions with other multi-server preprocessing PIR schemes. **Right**: Graph comparing the online bottleneck of our doubly efficient constructions with other multi-server preprocessing PIR schemes.

Theorem 1.1 (Informal: equal scaling setting)**.** *There exists an information-theoretic preprocessing S-server PIR scheme where each query incurs $\widetilde{O}(n^{O(1/\log S)})$ per-server bandwidth and computation, and $\widetilde{O}(S \cdot n^{O(1/\log S)})$ client computation, while requiring $\widetilde{O}(n^{1+O(1/\log S)})$ per-server space.*

Theorem 1.2 (Informal: minimizing bandwidth subject to scalable)**.** *There is an information-theoretic preprocessing S-server PIR scheme that achieves $n^{\widetilde{O}(1/S)}$ bandwidth and $n^{O(1/\log\log S)}$ computation per query, while requiring $n^{1+O(\log\log\log S/\log\log S)}$ server space.*

Theorem 1.3 (Informal: minimizing bottleneck cost subject to scalable)**.** *There is an information-theoretic preprocessing S-server PIR scheme that achieves $n^{\widetilde{O}(1/S^{1-1/\omega(1)})}$ bandwidth and computation per query while requiring $n^{1+\widetilde{O}(1/S^{1-1/\omega(1)})+O(\log\log n\cdot\omega(1)/\log S)}$ server space. In the above, we abuse the notation $\omega(1)$ to mean an arbitrarily small super-constant function in n.*

Relation to doubly-efficient. In the literature [LMW23, LLFP24], it is customary to use the term *doubly efficient* to describe a PIR scheme that achieves $n^{o(1)}$ bandwidth and computation per query, and $n^{1+o(1)}$ server space where $o(1)$ denotes any function that goes to 0 as n goes to infinity. Therefore, in Theorem 1.1 and Theorem 1.2, we achieve the notion of doubly efficient as long as S is a super-constant function in n. In Theorem 1.3, we would satisfy doubly efficient as long as S is super-polylogarithmic in n.

Graphical illustration. To aid understanding, we plot the the above results (Theorems 1.1 to 1.3), in Figure 1. For each curve, the vertical dotted line shows when the corresponding scheme becomes doubly efficient. In this figure, we also compare with two concurrent works:

- The first construction of Lazzaretti et al. [LLFP24, Theorem 4.1] scheme is shown in Figure 1 as the blue ♦, meaning that they are a special case of our Theorem 1.3. The second construction of Lazzaretti et al. [LLFP24, Theorem 5.2] scheme is shown in Figure 1 as the purple ♦, meaning that our construction in Theorem 1.1 can achieve lower bandwidth and online bottleneck cost using fewer servers.
- The scheme of Singh et al. [SWZ24] is shown as the dashed black curves in Figure 1. As mentioned earlier, their scheme does not achieve scalability in terms of computation per query, and this is why the dashed black curve in Figure 1 (right) is a horizontal line. While all other schemes in Figure 1 adopt server-side preprocessing, Singh et al. adopts client-side preprocessing where each client must

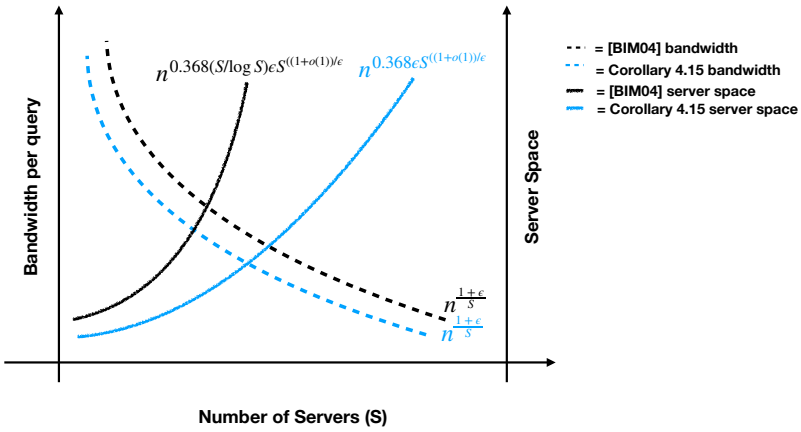

Figure 2: Graph comparing the bandwidth and space of our polynomial space setting scheme with [BIM04].

separately participate in a preprocessing protocol with the servers during a subscription phase. As a result, Singh et al. removes the need for additional server space, at the cost of requiring approximately $n^{1/2}$ space on each client.

1.2 Additional Results and Contributions

Improved results for the poly-space setting. While our paper focuses on scalable families of PIR schemes, the previous literature on preprocessing PIR [BIM04, WY05] instead asked what is the minimal bandwidth we can achieve subject to polynomial server space. We refer to this setting as the *poly-space setting*.

Our techniques for achieving scalable PIR also enable us to advance the state of the art in the poly-space setting. Specifically, we can improve the server space of Beimel et al. by a polynomial factor while approximately matching their bandwidth and computation, as stated in the following theorem.

Theorem 1.4 (Informal: the poly-space setting)**.** *For sufficiently large S, n, and sufficiently small $\epsilon > 0$, there exists an S-server PIR scheme where each query incurs $\widetilde{O}(n^{(1+\epsilon)/S})$ per-server bandwidth and computation, and $\widetilde{O}(n^{(1+\epsilon)/S} \cdot S)$ client computation, while requiring $n^{0.368\epsilon S^{(1+o(1))/\epsilon}}$ per-server space.*

In Figure 2, we compare our scheme in the poly-space setting with that of Beimel et al. [BIM04]. Specifically, their server space is an $S/\log S$-factor worse *in the exponent*. Further, with the same framework and different choice of parameters, we can also match the scheme with polylogarithmically many servers of Beimel et al. [BIM04] (see their Theorem 4.9 and our Appendix C). In Table 2, we compare the concrete efficiency of our schemes in Theorem 1.1, Theorem 1.4 and the scheme of [BIM04, Theorem 4.3].

A generic balancing technique. In all schemes mentioned so far, the download bandwidth is significantly larger than the upload bandwidth. We devise a generic technique for balancing these two costs, which reduces the total bandwidth at the cost of increased server computation. Our new balancing theorem differs in nature from the standard balancing theorem from prior literature [CGKS95] — the prior version focuses on the opposite scenario, where the upload bandwidth exceeds the download bandwidth.

Our balancing theorem is remotely related to a balancing technique described in the elegant work of Woodruff and Yekhanin [WY05] The main difference is that their balancing technique is tightly coupled with their specific PIR construction, and does not easily generalize. By contrast, we devise novel techniques and derive a balancing theorem that applies to *any* PIR scheme.

Table 1: **Comparison of concrete efficiency of our schemes with [BIM04, Theorem 4.3]**. The numbers represent the exponent of n ignoring additive $o(1)$ factors, e.g., 0.75 means $n^{0.75+o(1)}$.

S	Theorem 1.1		Theorem 1.4		[BIM04, Theorem 4.3]	
	Comm./Work	Storage	Comm./Work	Storage	Comm./Work	Storage
2	0.6824	1.6824	0.75	1.7735	0.75	1.7735
4	0.4541	1.4541	0.375	3.7832	0.375	5.4874
6	0.4063	1.4063	0.25	8.0230	0.25	14.0940
10	0.3258	1.3258	0.15	19.9255	0.15	51.5976
16	0.2725	1.2725	0.09	47.8135	0.09	179.0431

As an implication of this new balancing technique, we can obtain an improved 2-server PIR scheme as stated in the following theorem:

Theorem 1.5 (Informal: improved 2-server PIR in the poly-space setting)**.** *Let $\epsilon \in (0,1)$ be an arbitrarily small constant. There exists an information-theoretic 2-server PIR scheme with $n^{(1+\epsilon)/3}$ bandwidth and $n^{(2+\epsilon)/3}$ computation per query, while requiring a polynomial amount of server space.*

Theorem 1.5 almost matches the $O(n^{1/3})$ bandwidth achieved by Woodruff and Yekhanin [WY05], but we significantly improve their computation cost, from $n/\mathsf{poly}\log n$ to $n^{(2+\epsilon)/3}$. There is also evidence indicating potential barriers to further bandwidth improvements. Specifically, Razborov and Yakhanin showed that $n^{1/3}$ bandwidth is optimal for a natural class of bilinear and group-based 2-server PIR schemes [RY06]. To date, with the exception of Dvir and Gopi [DG16], all known 2-server PIR schemes (in all settings) satisfy this natural characterization, including this paper.

2 Technical Roadmap

2.1 Equal Scaling in All Dimensions from Multiplicity Codes

We first construct a family of multi-server PIR schemes from *multiplicity codes* and show how to parameterize the construction to get a PIR scheme where the bandwidth, server computation and space scale at the same rate. This scheme is doubly efficient when the number of servers is superconstant.

Background: multiplicity codes. Multiplicity codes, introduced by Kopparty et al. [KSY14], are a family of locally correctable codes (hence, also locally decodable codes) based on evaluations of polynomials and their derivatives. Multiplicity codes use *Hasse derivatives* which do not vanish even within a small field. A codeword for multiplicity code is obtained by evaluating an m-variate polynomial $F \in \mathbb{F}_q[X_1, \ldots, X_m]$ (where q is a prime or a prime power) of total degree at most d, along with all its derivatives of order $< t$, at all points in $\mathbb{F}_m^q$. In the context of locally decodable codes, the desired properties are for it to be locally decodable with a *low number of queries*, and for it to have *high rate* and *high minimum distance*. A code is k-query locally decodable if there is an algorithm that given a codeword, can recover the i-th bit of an n-bit message for any $i \in [\![n]\!]$[1] by querying the codeword at k locations (for multiplicity codes one location of the codeword is the evaluation of F and all its derivatives at one point in $\mathbb{F}_q^m$). The rate of a code is the ratio between the length of a message and a codeword. High rate means that the length of a codeword is close to the length of a message. Minimum distance is defined as the minimum number of locations two codewords differ in. High minimum distance means that the code can correct a large fraction of errors. For multiplicity

[1] For any $n \in \mathbb{N}$, $[\![n]\!]$ denotes the set $\{0, 1, \ldots, n-1\}$

codes, setting $m = O(1)$ and $t > m^2 = O(1)$ makes them n^ϵ-query locally decodable, for some constant ϵ. Moreover, these parameters achieve constant rate and constant minimum distance. This has been the regime of interest in prior works [KSY14, Y+12, Kop13] on multiplicity codes.

From multiplicity codes to PIR. Our goal is to construct a scalable family of S-server PIR scheme with global preprocessing using multiplicity codes of order-t evaluations of degree-d polynomials in m variables over $\mathbb{F}_q$. We encode the database DB into an m-variate polynomial F with degree d, i.e., we construct an injective mapping $E : [\![n]\!] \to \mathbb{F}_q^m$ and then using polynomial interpolation find F such that $F(E(i)) = \mathsf{DB}[i]$ for all $i \in [\![n]\!]$. Now suppose the client wants to query the database at index i such that $E(i) = \vec{u}$. The client chooses a random $\vec{v} \in \mathbb{F}_q^m$ and *distinct* and *non-zero* field elements $\lambda_0, \ldots, \lambda_{S-1}$ and sends $\vec{u} + \lambda_s \vec{v}$ to each server s where $s \in [\![S]\!]$. Each server returns to the client the value of F and all the Hasse derivatives up to order $t-1$ evaluated at the point it received.

Define $f(\lambda) = F(\vec{u}+\lambda\vec{v})$. Now, the client can compute the Hasse derivates of f up to order $t-1$ denoted from the servers' answers using a modified version of the chain rule. Then, using *Hermite interpolation of Hasse derivatives* [Has36, BGKM22], the client can recover $f(0) = F(\vec{u}) = F(E(i)) = \mathsf{DB}[i]$.

In order for the above PIR construction to work, certain constraints involving the parameters m, q, d, t, S need to be satified. We list them below.

- For the encoding E to exist, we need $q^m \geq n$.
- For the polynomial F to interpolate DB, F needs at least n monomials. This entails that $\binom{m+d}{m} \geq n$.
- To make sure S distinct non-zero points are in $\mathbb{F}_q$, we need $q \geq S + 1$.
- For the Hermite interpolation to work, we need that $t \cdot S > d$.

If the multiplicity code is S-query local, and satisfies these constraints, the construction that we outline above is an S-server PIR. The rate of the multiplicity code determines the server storage, i.e., the server storage will be n times the inverse of the rate. The minimum distance of the multiplicity code does not affect any parameter of the PIR because we do not need error correction.

If we directly use parameters for multiplicity codes similar to those used in the coding theory literature (e.g., [KSY14, Kop13, Y+12]), i.e., set $m = O(1)$ and $t = O(1)$, we would get an $\Omega(n^\epsilon)$-server PIR for any constant $\epsilon > 0$. As a PIR scheme, the number of servers is too large and thus this parameter regime is of little interest. Moreover, since we can accept polynomial server storage and do not care about error correction, we can use multiplicity codes with low rate and minimum distance close to zero. Therefore, PIR asks for a different parameter regime than the standard coding theory literature.

In order to set the parameters, q, m, d, t let us first describe how the performance of the PIR construction is affected by these parameters (we show the detailed calculation of these parameters in Section 4.2).

- The upload bandwidth is $O(m \log q)$, while the download bandwidth is $O\left(\binom{m+t-1}{t-1} \log q\right)$. The latter is larger since $t \geq 1$ and therefore dominates.
- The server computation is $O\left(\binom{m+t-1}{t-1} \log q\right)$.
- The client computation is $O\left(S \cdot \binom{m+t-1}{t-1} \cdot m \cdot \mathsf{poly} \log q\right) + O(\mathsf{poly}(d, \log q))$. The first term comes from the client's chain rule computation and the latter from the Hermite interpolation.
- The server storage is $O\left(\binom{m+t-1}{t-1} \cdot q^m \log q\right)$.
- The preprocessing time is $O\left(\binom{m+t-1}{t-1} \cdot q^m \cdot m \cdot \mathsf{poly} \log q\right)$.

Parameter setting for equal scaling. We now set our parameters for this construction making server space, computation and bandwidth at equal rates as S grows.

- Since q^m appears in both storage and preprocessing time, and we have the constraint $q^m \geq n$, we will set $m = \lceil \log n / \log q \rceil$, the smallest possible value of m satisfying this inequality.
- For setting d, we will again choose smallest possible value that we can show satisfies $\binom{m+d}{m} \geq n$. We want to minimize this because $\mathsf{poly}(d, q)$ appears in the client computation time, and we want to reduce it.

  Note that $\binom{m+d}{m} \geq ((m+d)/m)^m$ and we have set m such that $q^m \geq n$. So, it suffices to satisfy $(m+d)/m \geq q$. We choose $d = (q-1) \cdot m$.
- Now we set q, t. We want to minimize t because $\binom{m+t-1}{m}$ grows as t grows (after we fix m). We need to satisfy the constraint $t \cdot S > d = (q-1) \cdot m$, i.e., $t > ((q-1) \cdot m)/S$. Now, the minimum value of q is $S+1$ because we need S non-zero distinct field elements $\lambda_0, \ldots, \lambda_{S-1}$ to exist. For simplicity, we assume $S+1$ is a prime or prime power for now, and set $q = S+1$ because minimizing q minimizes S. Then, we set $t = m+1$, which is the minimum value of t satisfying the aforementioned constraint.

Formally, with precise calculations where we do not assume S is a prime or a prime power, we get the following theorem.

Theorem 2.1. *For any S, there exists an S-server PIR scheme which achieves $O(n^{2/(\log S - 1)} \log S)$ per-server bandwidth, $O(n^{2/(\log S - 1)} \log S)$ per-server computation and $O(n^{2/(\log S - 1)} S \log n \cdot \mathsf{poly} \log S)$ client computation per query, with $O(n^{1+2/(\log S - 1)} \cdot S \cdot \mathsf{poly} \log n)$ preprocessing time and server storage.*

The details are in Section 4.3, and a simplified version of the theorem is stated in Theorem 1.1. Notice that all the per server parameters decrease with the increase in the number of servers, i.e., the scheme is scalable. Further, as long as $S = \omega(1)$, our construction is doubly efficient.

2.2 Minimizing Bandwidth Subject to Scalability

Many prior works on PIR [Cha04, CMS99, DG16, LP23, GZS24, LP22] viewed bandwidth as the primary performance metric. In the equal-scaling scheme of Section 2.1, the bandwidth scales as $n^{O(1/\log S)}$. In comparison, prior works [BIM04] showed that if we forgo scalability in the server space, the bandwidth can scale as fast as $n^{O(1/S)}$. Therefore, a meaningful question is how to minimize the bandwidth while subject to scalability.

Unfortunately, relying solely on multiplicity codes appears to limit us to a bandwidth of $n^{O(1/\log S)}$ if we require scalability in all dimensions. Specifically, there is a direct tradeoff between the bandwidth and server space, such that further reducing bandwidth would lead to a super-polynomial blowup in server space. To achieve further bandwidth reduction, we need to make m larger, which in turn allows for smaller choices of d and t. As a result, the dominant term in the bandwidth expression, $\binom{m+t-1}{t-1}$, decreases. Unfortunately, increasing m beyond $\log n / \log q$ would cause the server space (related to q^m) to become super-polynomial

To overcome this dilemma, we turn to polynomial preprocessing algorithms. Intuitively, these algorithms make more efficient use of the server space while still preserving the ability to perform fast evaluation by storing some specialized data structures whose size is much smaller than q^m. This way we can set m to be larger without making the server space superpolynomial in n.

Here, we specifically use the polynomial preprocessing algorithm from [BGG+24] which has the following guarantee: Consider a m-variate polynomial $F \in \mathbb{F}_q[X_1, \ldots, X_m]$ of *individual degree* d' — this means total degree $d \leq m \cdot d'$:

- There is an algorithm Preprocess that takes as input the polynomial F, runs in time $O((16 \cdot d' \cdot (\log d' + \log\log q))^m \cdot \mathsf{poly}\log(m, d', q)$ and produces a data structure $\widetilde{F}$ of size $O((16 \cdot d' \cdot (\log d' + \log\log q))^m \cdot \mathsf{poly}\log(m, d', q)$.
- Given any $\vec{x} \in \mathbb{F}_q^m$, $\widetilde{F}(\vec{x}) = F(\vec{x})$, evaluating $\widetilde{F}$ takes time $O(16^m \cdot \mathsf{poly}\log(m, d', q))$.

The servers use the above polynomial preprocessing algorithm to preprocess F and its Hasse derivatives and store the data structures. When the query is made the servers use the stored data structures to compute their answers.

We set the parameters as follows: $m \approx \log n / \log\log q, q = S + 1, d' \approx \log q$ (this makes total degree $d \approx m \log q$), $t \approx m \log q / S$, the storage is not superpolynomial in n for superconstant S. It is easy to see that these parameters satisfy the constraints. Further m is much larger than t, so $\binom{m+t-1}{t-1} \approx (m/t)^t \approx (S/\log S)^{\log n \cdot \log S/(S \log\log S)} \approx n^{\widetilde{O}(1/S)}$. Therefore, the bandwidth scales as $n^{\widetilde{O}(1/S)}$. Moreover, 16^m is $n^{O(1/\log\log S)}$, $(d')^m$ is roughly n, $(\log d' + \log\log q)^m$ is $n^{O(\log\log\log S/\log\log S)}$. Thus the total storage is $n^{1+O(\log\log\log S/\log\log S)}$. Moreover, the server computation is dominated by the $16^m \cdot \binom{m+t-1}{t-1}$ term becomes $n^{\tilde{O}(1/S)+O(1/\log\log S)}$. As already discussed earlier, the bandwidth scales as $n^{\widetilde{O}(1/S)}$. Hence, the construction is also doubly efficient for any S superconstant in n.

Above, we used rough approximations in several places as indicated by the $\approx$ signs. With precise rounding and exact inequalities, we get the following theorem in Section 5.1.

Theorem 2.2. *There is an S-server PIR scheme which achieves the following parameters.*

- *per-server bandwidth* $n^{O\left(\frac{\log^2 S}{S\log\log S}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n, S)$,
- *per-server computation* $n^{O\left(\frac{\log^2 S}{S\log\log S}\right)+O\left(\frac{1}{\log\log S}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n, S)$,
- *client computation* $n^{O\left(\frac{\log^2 S}{S\log\log S}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n, S)$,
- *per-server preprocessing time and storage*

$$n^{1+O\left(\frac{\log^2 S}{S\log\log S}\right)+O\left(\frac{\log\log\log S}{\log\log S}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n, S)\ .$$

A simpler version of this theorem was stated in Theorem 1.2.

Details on the polynomial data structure. We briefly sketch the polynomial preprocessing algorithm that we use here. This algorithm in [BGG+24] which is an improvement on the fast polynomial evaluation algorithm from [KU08] that has been previously used in PIR constructions (e.g., [LMW23]).

Let F be a m-variate polynomial with *individual degree* d' over $\mathbb{Z}_q$. Given $\vec{x} \in \mathbb{Z}_q^m$, we want to evaluate $F(\vec{x})$ fast. It is easy to see that since F has at most $(d'+1)^m$ monomials and the maximum value of a monomial is $(q-1) \cdot (q-1)^{md'}$, the maximum value of F over $\mathbb{Z}$ is $M = (q-1)^{md'+1} \cdot (d'+1)^m$. In the algorithm in [KU08] the idea is to compute primes $p_1, p_2, \ldots, p_k : \prod_{j \in [k]} p_j > M$ where $p_1 < p_2 < \ldots < p_k$. We know that we can find primes such that $p_k = O(\log M) = O(md' \log q)$. Then the algorithm stores all evaluations of $F(\vec{x}) \bmod p_j$ for all $\vec{x} \in \mathbb{Z}_{p_j}^m$ for all $j \in [k]$. In the online phase in order to compute $F(\vec{x})$, the algorithm looks up $F(\vec{x}) \bmod p_j$ for all $j \in [k]$ and uses the Chinese Remainder Theorem to compute $F(\vec{x})$ from these values. Storing all the evaluations of $F(\vec{x}) \bmod p_j$ for all $\vec{x} \in \mathbb{Z}_{p_j}^m$ for all $j \in [k]$ requires space $O(p_j^m) = O((md' \log q)^m)$. By using recursion to instead compute the values of $F(\vec{x}) \bmod p_j$ by

decomposing p_j into smaller primes can make the $(\log q)^m$ factor reduce to $(\log \log q)^m$, but does not get rid of the m^m factor.

In [BGG+24], the idea is to instead find primes $p_1, p_2, \ldots, p_k : \prod_{j\in[k]} p_j^m > M$. Here, we can find these primes such that $p_j = O((\log M)/m) = O(d' \log q)$. Now, if we store all the evaluations $F(\vec{x}) \bmod p_j^m$ for all $j \in [k]$, for all $\vec{x} \in \mathbb{Z}_{p_j^m}^m$, the total storage is $O(p_j^{m^2})$, which is significantly worse than above! So, the key observation in [BGG+24] is that over $\mathbb{Z}_{p_j^m}$, the evaluation of an m-variate polynomial F at a point $\vec{x}$ can be derived from the evaluations of the Hasse derivatives of F upto order m at another point $\vec{y}$ such that the co-ordinates of $\vec{x} - \vec{y}$ are multiples of p_j. So, it just suffices to compute the evaluations of F and its Hasse derivatives upto order m at a certain subset of $\mathbb{Z}_{p_j^m}^m$ whose size is p_j^m. This helps us get rid of the m^m factor from the storage. It however turns out, just doing this is not yet enough for the result we want. We need to again recursively compute $F(\vec{y}) \bmod p_j^m$ by again decomposing p_j into even smaller primes. That suffices to get the parameters we want for our PIR construction.

2.3 Minimizing Bottleneck Cost Subject to Scalability

In Section 2.2, while the bandwidth scales as $n^{\widetilde{O}(1/S)}$, the online server computation is the online bottleneck and scales only as $n^{\tilde{O}(1/S)+O(1/\log\log S)}$. The server computation is also a part of the online bottleneck along with the bandwidth. So it is another natural question to minimize the online bottleneck cost subject to the construction being scalable.

To minimize the online bottleneck cost, the bandwidth and the server computation need to scale at a similar rate asymptotically. In the construction in Section 2.2 this was not possible because the server computation time included the time needed for evaluation of the polynomial data structure which was significantly larger than the bandwidth. To get around that, we instead need to use a polynomial preprocessing algorithm that has much faster evaluation at the cost of using more space.

So, we turn to the polynomial preprocessing algorithm from [KU08] which trades time off for more space compared to the algorithm in [BGG+24]. Concretely, for the algorithm in [KU08], the storage is $(md'(\log m + \log d' + \log\log q))^m \cdot \mathsf{poly}\log(m, d', \log q)$ and the evaluation time is $\mathsf{poly}\log(m, d', \log q)$. Since the online time is not exponential in m using this polynomial preprocessing algorithm would allow us to set parameters such that server computation time scales at a similar rate than the bandwidth. This was not possible when we used the scheme from [BGG+24] because the server computation had a 16^m term and $m \geq \log n/\log q$.

Here we choose $q = S + 1$ and $m \approx \log n \cdot \omega(1)/\log q$ where $\omega(1)$ is some superconstant function in n and it is in $o(\log n)$. We set $d' \approx S^{1/\omega(1)}$ which ensures $\binom{m+m\cdot d'}{m} \geq n$ and $t = \theta(m \cdot d'/S)$ which ensures $t \cdot S > m \cdot d'$

This makes $\binom{m+t-1}{t-1} \approx (m/t)^t = O(S^{1-1/\omega(1)})^{\log n\cdot\omega(1)/S^{1-\omega(1)}} \approx n^{\widetilde{O}(1/S^{1-\omega(1)})}$. This binomial coefficient is the dominating term in bandwidth and computation, and therefore the online bottleneck scales as $n^{\widetilde{O}(1/S^{1-\omega(1)})}$. Furthermore, $m^m \approx n^{O(\log\log n\cdot\omega(1)/\log S)}$, $(d')^m \approx n$ and $(\log d' + \log m + \log\log q) \approx n^{O(\log\log n\cdot\omega(1)/\log S)}$. The term $n^{O(\log\log n\cdot\omega(1)/\log S)}$ becomes $n^{o(1)}$ when S is superpolylogarithmic in n, making the storage $n^{1+o(1)}$.

Above, we used rough approximations in several places as indicated by the $\approx$ signs. With precise rounding and exact inequalities, we get the following theorem in Section 5.2.

Theorem 2.3. *Let $\omega(1)$ be any superconstant function such that it is in $o(\log n)$. There is an S-server PIR scheme which achieves the following parameters.*

- *per-server bandwidth* $n^{\widetilde{O}\left(\frac{1}{S^{1/\omega(1)}}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n) \cdot \log S$,
- *per-server computation* $n^{\widetilde{O}\left(\frac{1}{S^{1/\omega(1)}}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n, S)$,

- *client computation* $n^{\widetilde{O}\left(\frac{1}{S^{1/\omega(1)}}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n,S)$,
- *per-server preprocessing time and storage*

$$n^{1+\widetilde{O}\left(\frac{1}{S^{1/\omega(1)}}\right)+O\left(\frac{\log S}{\log n}\right)+O\left(\frac{\log\log n\cdot\omega(1)}{\log S}\right)} \cdot \mathsf{poly}\log(n,S)\ .$$

A simpler version of this theorem was stated in Theorem 1.3.

2.4 Additional Results for the Poly-Space Setting

We next consider the setting considered in earlier works [BIM04, WY05] where we focus on optimizing the bandwidth while allowing the server space to grow with respect to S as long as the server space is still polynomially bounded.

We slightly optimize our construction from Section 2.1 to improve the parameters for the PIR scheme. Compared to our scalable PIR family construction, here we choose a different kind of polynomial F than standard multiplicity code. Specifically we want F to be homogenous and *multi-linear*, i.e., all monomials in F have d variables and each variable has exponent one. Our encoding works as follows: we construct an injective mapping E which maps each element in $[\![n]\!]$ to an element of $\{0,1\}^m$ of hamming weight exactly d. For such a mapping to exist, we need $\binom{m}{d} \geq n$. Then we define an m-variate *multilinear homogeneous* polynomial F of degree d that satisfies $F(E(i)) = \mathsf{DB}[i]$. This makes our definition of F the same as that of Woodruff and Yekhanin [WY05]. We then encode the database with a multiplicity code for the polynomial F evaluated to order $< t$.

We adopt a new query and reconstruction algorithm that differs from both standard multiplicity code as well as Woodruff and Yekhanin [WY05]. Say the client wants to query the database at index i such that $E(i) = \vec{u}$. The client chooses a random $\vec{v} \in \mathbb{F}_q^m$. In the scalable construction the client chose S distinct non-zero field elements λ_s and sends $\vec{u} + \lambda_s\vec{v}$ to server s for $s \in [\![S]\!]$. In contrast, here we make the client choose S distinct field elements λ_s for $s \in [\![S]\!]$, but do not require all of them to be non-zero. The client sends $\lambda_s\vec{u} + \vec{v}$ to server S in this case. We can let one of the λ_s's be zero since it does not leak $\vec{u}$. This allows us to set the field size $q = S$ (assuming S is a prime, we describe the general case in Section 6). This optimization helps us reduce the field size by 1 (i.e., in the scalable constructed we needed $q \geq S + 1$), and since the server storage has a q^m multiplicative term, this helps us reduce the server storage by a polynomial factor.

The servers return the evaluations and all derivatives up to order $t-1$ of F at the point it received to the client. We define $f(\lambda) = F(\lambda\vec{u} + \vec{v})$. Again, the client can compute the Hasse derivates of f up to order $t-1$. Then, using *Hermite interpolation of Hasse derivatives* [Has36, BGKM22], the client can recover f. It turns out that the coefficient of λ^d in f is $\mathsf{DB}[i]$, which the client returns.

Here, in order to match the parameter setting in [BIM04, WY05], we need to set $q = S$, $m = O(\log n)$, $d = \theta m$ for $0 \leq \theta \leq 1/2$ (we show that it is possible to choose such m, d satisfying $\binom{m}{d} \geq n$) and $t = \lceil (d+1)/S \rceil$. In Table 2, we show how this construction improves on the server space compared to [BIM04]. Table 2 also shows how our optimization of choosing $\lambda_s\vec{u} + \vec{v}$ leads to savings in server space compared to the decoding strategy used for our scalable construction.

Remark 2.4 (Comparision with [WY05])**.** While our definition of F is same as that Woodruff and Yekhanin [WY05], because of their use of normal derivatives, they need to set $q > d$. Therefore, they cannot use a parameter regime like ours where $m = O(\log n), d = O(\log n)$. Instead, they use $q = S + 1, d = S, m = O(dn^{1/d})$. This makes q^m superpolynomial in n and hence their preprocessing cannot store all the evaluations and partial derivatives of F at q^m points. So, they devise a different preprocessing strategy tailored to their construction.

Further, we introduce a new balancing trick and apply it to our construction in the polynomial server space setting to generalize this construction. Using the balancing trick we can achieve a scheme where the per-server bandwidth is $n^{(1+\epsilon)/(S+1)}$, and the per-server computation is $n^{(2+\epsilon)/(S+1)}$.

2.5 A New Balancing Technique

Finally, we give an overview of our new balancing technique. Recall that for the 2-server setting, Beimel et al. [BIM04] can achieve $n^{1/2+\varepsilon}$ bandwidth and computation per query, whereas Woodruff and Yekhanin [WY05] can compress the bandwidth to $n^{1/3+\varepsilon}$ but as a tradeoff, their server computation increases abruptly to $n/\mathsf{poly}\log n$ which is only slightly sublinear. An interesting question is whether we can further reduce the bandwidth while trading off computation in a more graceful manner. For example, for the special case of 2 servers, can we match the $n^{1/3+\varepsilon}$ bandwidth of Woodruff and Yekhanin but still achieve n^{δ} server computation for some constant $\delta \in (0,1)$?

To answer the above questions, we describe a new balancing trick that allows us to enable a smooth tradeoff curve between bandwidth and computation. This allows us to generalize the design space for multi-server PIR. Given a (preprocessing) PIR scheme whose upload and download bandwidths are asymmetric, we want to use a balancing trick to balance the two to minimize the bandwidth.

Naïve balancing. In Lemma 4.4 of Beimel et al. [BIM04], they cite a naïve balancing trick originally proposed by Chor et al. [CGKS95]. However, this naïve trick is tailored for the case when the original PIR scheme has more upload bandwidth than download bandwidth. The idea is as follows. Suppose we have a database of n bits. We can divide it into $B := n^{1-\mu}$ blocks each of length n^{μ} for some appropriate $\mu \in (0,1)$. Now, to retrieve some index $i \in [\![n]\!]$ of the database that lies in block $r := \lfloor i/n^{\mu} \rfloor$, we run a separate PIR instance to retrieve the $(i \mod n^{\mu})$-th bit of each block, treating each block as a separate database of n^{μ} size. Further, all blocks may share the same query vector; however, the server needs to send a separate response for each block. Therefore, if the original PIR scheme has $\alpha(n)$ upload bandwidth and $\beta(n)$ download bandwidth for a database of size n, then the balanced scheme would have $\alpha(n^{\mu})$ upload bandwidth, and $n^{1-\mu} \cdot \beta(n^{\mu})$ download bandwidth. This naïve balancing trick works the best if the original PIR scheme has higher upload bandwidth than download bandwidth.

Unfortunately, Beimel et al. [BIM04]'s preprocessing PIR scheme as well as our improved version in Section 6 have the opposite behavior: the upload bandwidth is asymptotically smaller than the download bandwidth. In this case, this naïve balancing trick cannot reduce the bandwidth. Take their 2-serer scheme as an example: recall that it achieves $n^{(1+\epsilon)/2}$ computation and bandwidth per query. Suppose we now divide the database into $B := n^{1/3}$ blocks each of size $n^{2/3}$. Applying this balancing trick, the new bandwidth and computation per query becomes $n^{(1+\epsilon)/3} \cdot n^{1/3} = n^{(2+\epsilon)/3}$ which is worse than before.

New balancing technique: first attempt. We propose a new balancing trick for the case when the original PIR has higher download bandwidth than upload bandwidth. The idea is still to divide the database into blocks. However, we now want to aggregate the answers for all blocks rather than the queries for all blocks to save the download bandwidth. To understand our idea, it helps to first think of the following flawed attempt. Suppose that all S servers use the same deterministic algorithm to answer queries. We will have the client send to each server an honestly constructed query for the relevant block r that contains the desired index, and for all non-relevant blocks, the client sends the *same* random query to all S servers. Each server computes the summation mod S of the answers of all blocks. Now, for each non-relevant block, all servers have the same answer, so they cancel out under summation mod S. Unfortunately, with this scheme, the client could only get the summation (mod S) of all answers for the relevant block r too, and it is not clear how the client can reconstruct the correct answer — specifically, to correctly recover the answer using the underlying PIR scheme, the client would need to know all S answers for the relevant block r. Likely for this

exact reason, Woodruff and Yekhanin [WY05] came up with their own non-blackbox balancing trick that is tightly coupled with their scheme. However, their particular instantiation requires a large field size and this is one reason why they cannot achieve tighter server space and preprocessing cost.

Our idea. Unlike Woodruff and Yekhanin [WY05], we salvage the above flawed attempt and devise a general balancing trick for any "natural" (preprocessing) PIR scheme whose upload bandwidth is smaller than download bandwidth. Specifically, we modify the this flawed approach such that all non-relevant cancel out while still ensuring that the client can recover all S answers for the relevant block r.

The intuition is as follows. The n-bit database is divided into $B = n^{1-\mu}$ blocks each of size n^μ for some appropriate $\mu \in (0, 1)$. Suppose the desired index i lies in the r-th block. Then, the client will send real queries denoted $Q_{r,0}, \ldots, Q_{r,S-1}$ for the relevant block r to the S servers, and for every non-relevant block $j \neq r$, it will send the same random query denoted $Q_{j,0}$ to all S servers. Each server will compute the answers to all blocks. For each block's answer, the server will XOR it into one of two slots, called slot 0 and slot 1 respectively. The client signals to each server which slot to encode each block's answer by sending the server a random bit per block. Our construction guarantees the following invariants:

1. For the relevant block r, at least one server XORs the answer into slot 0, and at least one server XORs the answer into slot 1. In our actual construction, we simply make server 0 XOR it in a random slot b_r, and make all other servers XOR it in slot $1 - b_r$.
2. For each non-relevant blocks $j \neq r$, all servers XOR the answer of block j in the same random slot b_j.

This construction allows the client to recover all S answers for the relevant block r. Specifically, let J_b be the set of non-relevant blocks chosen for slot $b \in \{0, 1\}$. Suppose for some server $s \in [\![S]\!]$, the relevant block r's answer is XOR'ed into slot 0. Then, server s's response is of this form:

$$\left(\bigoplus_{j \in J_0} \mathsf{ans}_j\right) \bigoplus \mathsf{ans}_{r,s}, \quad \bigoplus_{j \in J_1} \mathsf{ans}_j \tag{1}$$

In the above, $\mathsf{ans}_{r,s}$ denotes server s's answer (of the underlying PIR scheme) for the relevant block r. Further, for $j \neq r$, ans_j denotes the answer for block j of the underlying PIR scheme — since all S servers have the same answer for each non-relevant block $j \neq r$, we omit the server index in the notation ans_j. Now, suppose there exists another server s' who XORs the relevant block r's answer into slot 1, then its response to the client is of the form

$$\bigoplus_{j \in J_0} \mathsf{ans}_j, \quad \left(\bigoplus_{j \in J_1} \mathsf{ans}_j\right) \bigoplus \mathsf{ans}_{r,s'}, \tag{2}$$

Clearly, we can recover both $\mathsf{ans}_{r,s}$ and $\mathsf{ans}_{r,s'}$ from Equation (1) and Equation (2). Specifically, this can be done by XORing the two servers' answers for the each of the two slots. Generalizing this, as long as the above two conditions are satisfied, the client can recover all S answers (of the underlying PIR) for the r-th block, denoted $\{\mathsf{ans}_{r,s}\}_{s \in [\![S]\!]}$. It can now call the underlying PIR's reconstruction algorithm to reconstruct the answer for the relevant block r.

Applying the balancing trick. Suppose we start with a scheme with $n^{(1+\epsilon)/S}$ bandwidth and computation and $\mathsf{poly}(S)$ preprocessing cost and server space such as Beimel et al. [BIM04] or our new scheme with improved server space. We can apply this balancing trick by choosing $\mu = S \cdot \alpha$ for any $\alpha \in [1/(S + 1), 1/S]$. The resulting scheme will enjoy $O(n^{(1+\epsilon)\alpha} \log S)$ per-server bandwidth, $n^{1-(S-1-\epsilon)\alpha}$ per-server

computation, and the server space and preprocessing cost are still polynomially bounded. Specifically, if we take $\alpha = 1/(S+1)$, the upload and download bandwith will be balanced (up to $1+\epsilon$ factors in the exponent). In this case, the per-server bandwidth is minimized to $n^{(1+\epsilon)/(S+1)}$, and the per-server computation is $n^{(2+\epsilon)/(S+1)}$.

3 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 3.1 (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 $s \in [\![S]\!]$ 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 $i \in [\![n]\!]$, 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.
- $\mathbf{Answer}_s(\widetilde{\mathsf{DB}}_s, Q_s)$: given the encoded database $\widetilde{\mathsf{DB}}_s$ and a query message Q_s of server $s \in [\![S]\!]$, 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 $i \in [\![n]\!]$,

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

Security. Security requires that any individual server's view leaks nothing about the client's desired index. Formally, for any n, S, for any $i_1, i_2 \in [\![n]\!]$ and any $s \in [\![S]\!]$, 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.

Further, we say that a multi-server PIR scheme is *doubly efficient* if all these conditions hold.

- $\mathbf{Preproc}_s$ runs time $n^{1+o(1)}$ for all $s \in [\![S]\!]$ and its output is of size $n^{1+o(1)}$ i.e., the preprocessing time and the server storage is $n^{1+o(1)}$.
- The combined runtime of $\mathbf{Query}, \mathbf{Recons}$ is $n^{o(1)}$ i.e., the client computation per query is $n^{o(1)}$.
- The runtime of $\mathbf{Answer}_s$ is $n^{o(1)}$ i.e., the server computation per query is $n^{o(1)}$.
- The sum of the size of the output of $\mathbf{Query}$ and the maximum size of the output of $\mathbf{Answers}_s$ for $s \in [\![S]\!]$ is $n^{o(1)}$, i.e., the per server bandwidth per query is at most $n^{o(1)}$.

4 Preprocessing PIR from Multiplicity Codes

In this section, we introduce a framework to construct a family of server-side preprocessing PIR schemes from multiplicity codes. In Section 4.1, we give some preliminaries about multiplicity codes. In Section 4.2, we present our family of PIR schemes from multiplicity codes. The parameters of the PIR construction depends on the parameters of the multiplicity code. In Section 4.3, we show how to set the parameters to get a new scalable family of PIR schemes where the (per-server) bandwidth and computation per query as well as the server space scale with respect to the number of servers. This new scalable family also allows us to get a doubly efficient PIR scheme with only super-constant servers. In Section 6, we present some further optimizations to our construction in order to optimize the bandwidth in the polynomial space setting.

4.1 Preliminaries on Multiplicity Codes

We define $A_{k,m}$ to be the set of all vectors of non-negative integers of length m and 1-norm exactly k:

$$A_{k,m} = \{\vec{a} \in \mathbb{N}^m : \mathsf{wt}(\vec{a}) = k\}$$

where $\mathsf{wt}(\vec{a}) = \vec{a}_1 + \ldots + \vec{a}_m$ denotes the 1-norm of the vector $\vec{a}$. Let $A_{<k,m} := A_{0,m} \cup A_{1,m} \ldots \cup A_{k-1,m}$.

Further, for $d \in \mathbb{N}$, we define $A_{k,m,d}$ to be the set of all vectors of length m whose entries are in $\{0, 1, \ldots, d\}$ and have 1-norm exactly k:

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

We $A_{<k,m,d} := A_{0,m,d} \cup A_{1,m,d} \ldots \cup A_{k-1,m,d}$.

Given $\vec{a} := (\vec{a}_1, \ldots, \vec{a}_m) \in \mathbb{N}^m$, and a polynomial F, we define the partial derivative operator $\boldsymbol{\partial}^{\vec{a}}$ as:

$$\boldsymbol{\partial}^{\vec{a}} \circ F := \frac{\partial^{\mathsf{wt}(\vec{a})} F}{\partial X_1^{\vec{a}_1} \ldots \partial X_m^{\vec{a}_m}}$$

Henceforth, given a vector $\vec{X} := (\vec{X}_1, \ldots, \vec{X}_m)$ of variables and a vector $\vec{a} := (\vec{a}_1, \ldots, \vec{a}_m)$ of exponents, we use the following vector exponentiation notation:

$$\vec{X}^{\vec{a}} := \prod_{k=1}^{m} \vec{X}_k^{\vec{a}_k}$$

Definition 4.1 (Hasse derivatives)**.** For m-variate polynomial $F \in \mathbb{F}[X_1, \ldots, X_m]$ over the field $\mathbb{F}$, the Hasse derivative of f with respect to $\vec{a} = (\vec{a}_1, \ldots, \vec{a}_m) \in \mathbb{N}^m$ is defined as

$$\overline{\partial}^{\vec{a}} \circ F = \sum_{\vec{e}=(\vec{e}_1,\ldots,\vec{e}_m) \in \mathbb{N}^m} \prod_{i=1}^{m} \binom{\vec{a}_i}{\vec{e}_i} \cdot \mathsf{Coeff}_{X_1^{\vec{e}_1} \ldots X_m^{\vec{e}_m}}(F) \vec{X}^{\vec{a}-\vec{e}}$$

where $\mathsf{Coeff}_{X_1^{\vec{e}_1} \ldots X_m^{\vec{e}_m}}(F)$ denotes the coefficient of $X_1^{\vec{e}_1} \ldots X_m^{\vec{e}_m}$ in F.

Specifically, for a univariate degree-d polynomial $f(\lambda) = \sum_{k=0}^{d} c_k \cdot \lambda^k \in \mathbb{F}[\lambda]$, we omit the vector notation and denote its r-th Hasse derivative as $\overline{\partial}^{(r)} f(\lambda) = \overline{\partial}^{(\vec{r})} \circ f(\lambda) = \sum_{k=r}^{d} c_k \cdot \binom{k}{r} \lambda^{k-r}$. If the field $\mathbb{F}$ has characteristic 0, then $\overline{\partial}^{(r)} f(\lambda) = \frac{1}{r!} f^{(r)}(\lambda)$.

Chain Rule. Given a univariate polynomial $g \in \mathbb{F}[\lambda]$, 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 4.2 (Chain rule for normal derivatives)**.** *For m-variate polynomial $f(X_1, \ldots, X_m)$ 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*

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

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:

Lemma 4.3 (Chain rule for Hasse derivatives)**.** *For m-variate polynomial $f(X_1, \ldots, X_m)$ over field $\mathbb{F}$, let $g(\lambda) = f(\vec{u} + \lambda\vec{v})$ be a univariate polynomial in λ, we have*

$$\overline{\partial}^{(k)} g(\lambda) = \sum_{\vec{a} = (\vec{a}_1, \ldots, \vec{a}_m) \in A_{k,m}} \overline{\partial}^{\vec{a}} \circ f(\vec{u} + \lambda\vec{v}) \cdot \vec{v}^{\vec{a}}$$

Specifically, when f is a multilinear polynomial (a multivariate polynomial with individual degree 1 in each variable), clearly $\overline{\partial}^{\vec{a}} \circ f = \boldsymbol{\partial}^{\vec{a}} \circ f$ for any $\vec{a} \in A_{k,m,1}$, therefore,

$$\overline{\partial}^{(k)} g(\lambda) = \sum_{\vec{a} = (\vec{a}_1, \ldots, \vec{a}_m) \in A_{k,m,1}} \boldsymbol{\partial}^{\vec{a}} \circ f(\vec{u} + \lambda\vec{v}) \cdot \vec{v}^{\vec{a}}$$

Hermite Interpolation. It is well-known that for a polynomial f over field with characteristic 0, given sufficiently many evaluations of f and higher-order (normal) derivatives of f at some points, it suffices to reconstruct f, known as Hermite interpolation. The same holds for Hasse derivatives, the difference is that Hasse derivative version works not only on field with characteristic 0, but also arbitrary finite fields:

Lemma 4.4 (Hermite interpolation of Hasse derivatives [Has36, BGKM22])**.** *Let f be an univariate polynomial of degree d over finite field $\mathbb{F}$, and m positive integers $e_1, \ldots, e_m$ such that $e_1 + \cdots + e_m > d$. Let $\alpha_1, \ldots, \alpha_m$ be distinct elements in $\mathbb{F}$. For all $i \in [m]$ and $j \in [e_i]$, let $\overline{\partial}^{(j-1)} f(\alpha_i) = y_{i,j}$. Then, the coefficients of f can be recovered from $\{(\alpha_i, j, y_{i,j})\}_{i \in [m], j \in [e_i]}$ in time $\mathsf{poly}(d, \log |\mathbb{F}|)$.*

Multiplicity codes. Finally, we come to the definition of multiplicity codes. We use the definition from [KSY14].

Definition 4.5 (Multiplicity Code [KSY14])**.** Let t, d, m be non-negative integers and let q be a prime power. Let $\Sigma = \mathbb{F}_q^{\binom{m+t-1}{m}} = \mathbb{F}_q^{\{\vec{a} \in A_{<t,m}\}}$. For $F(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$ we define the order t evaluation of F at $\vec{x}$ to be the vector $(\overline{\partial}^{\vec{a}} \circ F(\vec{x}))_{\vec{a} \in A_{<t,m}}$. The multiplicity code of order-t evaluations of degree-d polynomials in m variables over $\mathbb{F}_q$ is defined as follows. The code is over alphabet Σ and has length q^m, where the coordinates are indexed by elements of $\mathbb{F}_q^m$. For each polynomial $F(\vec{X}) \in \mathbb{F}_q[X_1, \ldots, X_m]$ with $\deg(F) \leq d$, there is a codeword given by

$$\mathbf{Encode}_{t,d,m,q}(F) = \left((\overline{\partial}^{\vec{a}} \circ F(\vec{x}))_{\vec{a} \in A_{<t,m}} \right)_{\vec{x} \in \mathbb{F}_q^m}.$$

We will prove the following claim related to multiplicity codes of order-t evaluations of degree-d polynomials in m variables over $\mathbb{F}_q$ that will be useful later.

Claim 4.6. Let $t, d, m > 0$ and $S > 1$ be integers. Let q be a prime or prime power and $\vec{u}, \vec{v} \in \mathbb{F}_q^m$. Let $\lambda_0, \lambda_1, \ldots, \lambda_{S-1}$ be distinct nonzero elements of $\mathbb{F}_q$. Let $\vec{z}_s = \vec{u} + \lambda_s \vec{v}$ for $s \in [\![S]\!]$. For a polynomial $F \in \mathbb{F}_q[X_1, \ldots, X_m]$ such that $\deg(F) \leq d$, given the evaluation of $\overline{\partial}^{\vec{a}} \circ F(\vec{z}_s)$ for all $\vec{a} \in A_{<t,m}$, $s \in S$, we can recover $F(\vec{u})$ if $S \cdot t > d$.

Proof. Let $f(\lambda) = F(\vec{u} + \lambda \vec{v})$. Using the chain rule for Hasse derivatives, we have that for all $s \in [\![S]\!]$ and $0 \leq k < t$

$$\overline{\partial}^{(k)} f(\lambda_s) = \sum_{\vec{a} \in A_{k,m}} \overline{\partial}^{\vec{a}} \circ F(\vec{z}_s) \cdot \vec{v}^{\vec{a}} \,. \tag{3}$$

Note that given the evaluation of $\overline{\partial}^{\vec{a}} \circ F(\vec{z}_s)$ for all $\vec{a} \in A_{<t,m}$, $\vec{z}_s$ for $s \in [\![S]\!]$, we can compute the right hand side of the other expression when $0 \leq k < t$.

Now observe that $\deg(f) \leq \deg(F) \leq d$. So, it follows from Lemma 4.4 that we can reconstruct f by Hasse derivatives computed above as long as $S \cdot t > d$.

□

4.2 PIR Family from Multiplicity Codes

In this section, we present the construction of the family of PIR schemes from multiplicity codes. We start off by giving some intuition about how we use multiplicity codes to construct PIR schemes.

Our goal is to construct a scalable family of S-server PIR scheme with global preprocessing using multiplicity codes of order-t evaluations of degree-d polynomials in m variables over $\mathbb{F}_q$. We encode the database DB into an m-variate polynomial F with degree d, i.e., we construct an injective mapping $E : [\![n]\!] \to \mathbb{F}_q^m$ and then find such F using interpolation satisfying $F(E(i)) = \mathsf{DB}[i]$ for all $i \in [\![n]\!]$. Now suppose the client wants to query the database at index i such that $E(i) = \vec{u}$. The client chooses a random $\vec{v} \in \mathbb{F}_q^m$ and distinct non-zero field elements $\lambda_0, \ldots, \lambda_{S-1}$ and sends $\vec{u} + \lambda_s \vec{v}$ to server s for $s \in [\![S]\!]$. The server returns the evaluations and all derivatives up to order t of F at the point it received to the client. Using Claim 4.6, the client can recover $F(\vec{u}) = \mathsf{DB}[i]$.

In order to make the server's online computation efficient, we make the server store all possible evaluations and Hasse derivates of the polynomial F. To make the preprocessing phase of the server efficient, we do the following:

- In order to compute the polynomial F described above, we use the interpolation algorithm of Lin, Mook, and Wichs [LMW23, Lemma 2.2] for encoding a database of size at most q^m elements into a m-variate polynomial F over $\mathbb{F}_q$.
  - Given an **injective** map $E : [\![n]\!] \to \mathbb{F}_q^m$, there is an interpolation algorithm that takes $n \leq q^m$ values $\{y_i\}_{i \in [\![n]\!]}$, and recovers coefficients of a polynomial $F(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$ with individual degree at most d' in each variable such that $F(E(i)) = y_i$ for all $i \in [\![n]\!]$. Further, the algorithm runs in time $O((d')^m \cdot m \cdot \mathsf{poly} \log q)$. When we have an upper bound on total degree instead of total degree, individual degree cannot be more than $q - 1$, so in that case the algorithm runs in time $O(q^m \cdot m \cdot \mathsf{poly} \log q)$.
- In order to pre-compute the evaluation of $\overline{\partial}^{\vec{a}} \circ F$ for all $\vec{a} \in A_{\leq t,m}$, $\vec{(x)} \in \mathbb{F}_q^m$ we use the result from [KU11, Theorem 4.1] (stated below as Lemma 4.7), which states that given an m-variate polynomial F over prime field $\mathbb{F}_q$, we can simultaneously evaluate it at every point of $\mathbb{F}_q^m$ in almost linear time.

Lemma 4.7 ([KU11]). *There is a deterministic algorithm that takes coefficients of an m-variate polynomial F over finite field $\mathbb{F}_q$ (without loss of generality we may assume F has individual degree at most $q-1$ in each variable) as input then outputs $F(\vec{X})$ for all $\vec{X} \in \mathbb{F}_q^m$, and runs in time $O(q^m \cdot m \cdot \mathsf{poly}\log q)$.*

We give the formal description of our construction below.

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

- $\mathbf{Preproc}_s$: Encode database DB to m-variate polynomial F with total degree d. Concretely, we construct $E : [\![n]\!] \to \mathbb{F}_q^m$ be an *injective* index function, and recover F by interpolating on the set $\{\mathsf{DB}[i]\}_{i\in[\![n]\!]}$ using the techniques described by Lin et al. [LMW23].

  For each $\vec{a} \in A_{<t,m}$, we can use the preprocessing algorithm described in Lemma 4.7 to pre-compute evaluation of $\overline{\partial}^{\vec{a}} \circ F$ at any point $x \in \mathbb{F}_q^m$.
- $\mathbf{Query}$: Given query index i, the client uniformly generates $\vec{v} \in \mathbb{F}_q^m$, and sets $\vec{u} = E(i)$. Then it picks S distinct and **nonzero** elements in $\mathbb{F}_q$ called $\lambda_0, \ldots, \lambda_{S-1}$.

  For $s \in [\![S]\!]$, the client sets
  $$\vec{z}_s = \vec{u} + \lambda_s \vec{v}.$$
  The client sends $Q_s = \vec{z}_s$ to each server $s \in [\![S]\!]$.
- $\mathbf{Answer}_s$: The s-th server ($s \in \{0, 1, \ldots, S\}$) parses the message received from the client as a vector $\vec{z}_s$. For each $\vec{a} \in A_{<t,m}$, it sends
  $$\mathsf{ans}_{s,\vec{a}} = \underbrace{\overline{\partial}^{\vec{a}} \circ F(\vec{z}_s)}_{\text{precomputed during preproc}}$$
  back to the client.
- $\mathbf{Recons}$:
  1. Define univariate polynomial $f(\lambda) = F(\vec{u} + \lambda\vec{v})$, clearly $\vec{z}_s = \vec{u} + \lambda_s\vec{v} = f(\lambda_s)$. Given the responses of all servers, the client computes $\overline{\partial}^{(k)} f(\lambda_s)$ for all $s \in [\![S]\!]$ and $0 \le k < t$ by:
  $$\overline{\partial}^{(k)} f(\lambda_s) = \sum_{\vec{a}\in A_{k,m}} \overline{\partial}^{\vec{a}} \circ F(\vec{z}_s) \cdot \vec{v}^{\vec{a}} = \sum_{\vec{a}\in A_{k,m}} \mathsf{ans}_{s,\vec{a}} \cdot \vec{v}^{\vec{a}}$$
  2. Reconstruct f by its Hasse derivatives. Since F has degree d, f has degree at most d. For each $s \in [\![S]\!]$, the client has already known its k-th order Hasse derivatives at point λ_s for any $0 \le k < t$. So, if $S \cdot t > d$, the client can use Hermite interpolation (Lemma 4.4) to reconstruct the coefficients, and output $f(0) = F(\vec{u}) = \mathsf{DB}[i]$ as the answer.

For the above to work, certain constraints need to be satisfied. We list them below.

Constraints For database size n,

- To ensure that the *injective* mapping $E : [\![n]\!] \to \mathbb{F}_q^m$ exists, we need $q^m \ge n$.
- To ensure that a polynomial m-variate polynomial F with degree at most d interpolates n points, the number of monomials in F has to be at least n. Therefore, $\binom{m+d}{m} \ge n$.
- Since the client sends $\vec{u} + \lambda_s\vec{v}$ to server $s \in [\![S]\!]$, and since none of the λ_s can be zero (because that would leak $\vec{u}$, thus compromising privacy), the field must constant at least S non-zero elements. Therefore, $q \ge S + 1$.
- For reconstruction to work, we need $S \cdot t > d$.

4.2.1 Proof of Correctness

The correctness of PIR scheme immediately follows from Claim 4.6 above.

4.2.2 Proof of Security

Each server $s \in [\![S]\!]$ 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 $\lambda_s \neq 0$ and $\vec{v}$ is randomly sampled.

4.2.3 Efficiency

We now analyze the efficiency of our construction.

- **Bandwidth**: The upload bandwidth per server is $m \log q$ since the client sends an element in $\mathbb{F}_q^m$ to each server. Each server should return $|A_{<t,m}|$ elements in $\mathbb{F}_q$, and we have

$$|A_{<t,m}| = \binom{t+m-1}{m}.$$

  Thus the total per-server download bandwidth is $|A_{<t,m}| \log q = \binom{m+t-1}{m} \log q$.

- **Server computation**: Each server simply sends back $|A_{<t,m}|$ stored values, so the total computation is $O\left(\binom{m+t-1}{m} \log q\right)$.

- **Client computation**: For applying the chain rule, for each server s and $\vec{a} \in A_{<t,m}$, the client should do $O(m)$ multiplications in $\mathbb{F}_q$ (where each takes $O(\mathsf{poly} \log q)$ time). This takes time $O\left(S \cdot \binom{m+t-1}{m} \cdot m \cdot \mathsf{poly} \log q\right)$. The hermite interpolation takes time $\mathsf{poly}(d, \log q)$. Therefore, total client computation is $O\left(S \cdot \binom{m+t-1}{m} \cdot m \cdot \mathsf{poly} \log q\right)$ $O(\mathsf{poly}(d, \log q))$.

- **Server space**: Each server runs preprocessing algorithm in Lemma 4.7 for $|A_{<t,m}|$ polynomials. For each polynomial, it needs to store q^m elements in $\mathbb{F}_q$, thus in total it takes space

$$|A_{<t,m}| \cdot O(q^m \log q) = \binom{m+t-1}{m} \cdot q^m \cdot \log q$$

- **Preprocessing time**: Each server needs to interpolate the polynomial F which takes time $O(q^m \cdot m \cdot \mathsf{poly} \log q)$. For each $\vec{a} \in A_{<t,m}$ $\overline{\partial}^{\vec{a}} \circ f$ can be computed in time $O(q^m \cdot m \cdot \mathsf{poly} \log q)$. Therefore, the total time foe computing these polynomials is $|A_{<t,m}| \cdot O(q^m \cdot m \cdot \mathsf{poly} \log q)$. Further, each of these polynomials are to be evaluated in q^m points. Using Lemma 4.7, each evaluation takes time $O(q^m \cdot m \cdot \mathsf{poly} \log q)$.

  Therefore, the total preprocessing time is

$$O((|A_{<t,m}| \cdot O(q^m \cdot m \cdot \mathsf{poly} \log q)) = O\left(\binom{m+t-1}{m} \cdot q^m \cdot m \cdot \mathsf{poly} \log q\right).$$

Theorem 4.8. *Let $n, S, d, m, q, t \in \mathbb{N}$ satisfy the following constraints:*

- *q is a prime or a prime power with $q \geq S + 1$,*
- *$q^m \geq n$,*
- *$\binom{m+d}{m} \geq n$,*

- $S \cdot t > d$.

Then, there exists an S-server PIR scheme which achieves the following parameters.

- *per-server upload bandwidth* $m \log q$*, per-server download bandwidth* $\binom{m+t-1}{m} \log q$.
- *per-server computation* $O\left(\binom{m+t-1}{m} \log q\right)$.
- *client computation* $O\left(S \cdot \binom{m+t-1}{m} \cdot m \cdot \mathsf{poly} \log q\right) + O(\mathsf{poly}(d, \log q))$.
- *per-server storage* $\binom{m+t-1}{m} \cdot q^m \cdot \log q$.
- *per-server preprocessing time* $O\left(\binom{m+t-1}{m} \cdot q^m \cdot m \cdot \mathsf{poly} \log q\right)$.

4.3 Scaling Bandwidth, Computation and Space Equally

In this section, we show how to set the parameters for the construction in Section 4.2 to get a scalable family of PIR schemes where the (per-server) bandwidth and computation per query as well as the server space scale with respect to the number of servers. The scheme is same as described in section 4.2, with the following parameters.

- Let $S(n)$ be the number of servers, we pick $S^*(n) \leq S(n)$ to be the maximum integer such that $S^*(n)+1$ is a prime (by Bertrand's postulate, $S^*(n) \in [\lfloor S(n)/2 \rfloor, S(n)]$), then use only $S^*(n)$ servers, and ignore the other servers.
- We set $q = S^*(n) + 1$
- Set $m = \lceil \log n / \log q \rceil$ such that $q^m \geq n$.
- We set $d = (q-1) \cdot m$.
- We will set $t = m + 1$ which ensures $t \cdot S^*(n) > d$.

The constraints $q \geq S+1, q^m \geq n, t \cdot S > d$ can be seen to be satisfied from the above. Note that

$$\binom{m+d}{d} = \binom{(q-1)\cdot m}{m} \geq (q-1)^m \geq n \ .$$

Therefore, all the constraints are satisfied. Observe that since $t = m$, it follows that

$$\binom{m+t-1}{t-1} \leq \binom{2m}{m} \leq 2^{2m} \leq O(n^{2/\log q}) \ .$$

We now compute the performance parameters for the PIR scheme.

- **Bandwidth**: The bandwidth will be dominated by the download bandwidth. Thus, the total per-server bandwidth is bounded by $O(n^{2/\log q} \log q)$.
- **Server computation**: The total computation is that is, $O(n^{2/\log q} \log q)$.
- **Client computation**: Note that the first term dominates the client computation. So it is $O(S \cdot n^{2/\log q} \cdot \mathsf{poly} \log q \cdot \mathsf{poly} \log n)$.
- **Server space**: Since $q^m \leq n \cdot q$, we have that the total server space is $O(n^{1+2/\log q} \cdot q \cdot \log q)$.
- **Preprocessing time**: The preprocessing time is $O(n^{1+2/\log q} \cdot q \cdot \mathsf{poly} \log n)$

Notice that our parameterization guarantees $q > S(n)/2$ and $2/\log q \leq 2/(\log S(n) - 1)$. So, we have the following theorem.

Theorem 4.9. *For any S, there exists an S-server PIR scheme which achieves $O(n^{2/(\log S-1)} \log S)$ per-server bandwidth, $O(n^{2/(\log S-1)} \log S)$ per-server computation and $O(n^{2/(\log S-1)} S \log n \cdot \mathsf{poly}\log S)$ client computation per query, with $O(n^{1+2/(\log S-1)} \cdot S \cdot \mathsf{poly}\log n)$ preprocessing time and server storage.*

For constant number servers setting, our construction shows nontrivial results about tradeoff between preprocessing time and bandwidth.

If we choose $S(n) = \omega(1)$ to be any super-constant function (e.g. $S(n) = \log^*(n)$), then we have $2/\log q = 2/\log \Omega(S(n)) = o(1)$, thus $\binom{m+t-1}{t-1}$ is bounded by $n^{o(1)}$. Moreover, the polylogarithm factor can be absorbed by $n^{o(1)}$. Therefore, we get a doubly-efficient mutli-server PIR using a superconstant number of servers. In conclusion, we have:

Corollary 4.10. *For any $S(n) = \omega(1)$, there exists an S-server PIR scheme where each query incurs $n^{o(1)}$ per-server bandwidth and computation and $S \cdot n^{o(1)}$ client computation, while requiring $n^{1+o(1)}$ per-server preprocessing time and storage. Specifically, when $S(n) = n^{o(1)}$, the client computation is also bounded by $n^{o(1)}$.*

5 Multi-Server PIR Using Multiplicity Codes and Fast Polynomial Evaluation Data Structure

In this section, we present families of PIR schemes that use multiplicity codes in conjunction with fast polynomial evaluation algorithms. In our scheme in Section 4, our preprocessing involved storing the evaluation of the polynomials at all points. Now, we will use fast polynomial evaluation algorithms from [KU08, BGG+24] where the server does not pre-compute the evaluation of the polynomials at all points, and instead stores a data structure. In the online phase, the server uses this data structure to evaluate the polynomials at the desired point.

We will construct two incomparable families of PIR schemes using two different polynomial preprocessing algorithms. We will first use the following algorithm from [BGG+24].

Theorem 5.1 (Polynomial preprocessing theorem from [BGG+24])**.** *For a m-variate polynomial $f : \mathbb{F}_q^m \to \mathbb{F}_q$ with individual degree $\leq d'$, over some finite field $\mathbb{F}_q$, then there exists algorithms* PolyPreprocess, EvalPoly *such that*

- *The runtime of* $\mathsf{PolyPreprocess}(f)$ *is* $(16d'(\log d' + \log\log q)^m \cdot \mathsf{poly}(m, d', \log q)$
- *Let* $\widetilde{f} \leftarrow \mathsf{PolyPreprocess}(f)$*, for any* $\vec{x} \in \mathbb{F}_q^m$, $\mathsf{EvalPoly}(\widetilde{f}, \vec{x}) = f(\vec{x})$ *and the runtime of* $\mathsf{EvalPoly}(\widetilde{f}, \vec{x})$ *is* $16^m \cdot \mathsf{poly}(m, d', \log q)$.

Since [BGG+24] give an algorithm that does not have a separate preprocessing and online phase, for the sake of completeness we give a self-contained proof of this theorem in Appendix A.

Using this theorem will give us a family of PIR schemes where the bandwidth scales as $n^{\widetilde{O}(1/S)}$. This construction scales with respect to all parameters and minimizes bandwidth. Further, this construction is doubly efficient with a superconstant number of servers, As we discuss in Remark 5.5, we could not have achieved a construction that is doubly efficient with just super-constant number of servers and has similar bandwidth scaling using the polynomial preprocessing algorithms from [BGG+24].

We will then use the following fast polynomial evaluation algorithm from [KU08] to get a different scheme.

Theorem 5.2 (Polynomial preprocessing theorem from [KU08]). *For a m-variate polynomial $f : \mathbb{F}_q^m \to \mathbb{F}_q$ with individual degree $\leq d'$, over some finite field $\mathbb{F}_q$, then there exists algorithms* PolyPreprocess, EvalPoly *such that*

- *The runtime of* $\mathsf{PolyPreprocess}(f)$ *is* $(md'(\log d' + \log m + \log\log q)^m \cdot \mathsf{poly}(m, d', \log q)$
- *Let* $\widetilde{f} \leftarrow \mathsf{PolyPreprocess}(f)$*, for any* $\vec{x} \in \mathbb{F}_q^m$, $\mathsf{EvalPoly}(\widetilde{f}, \vec{x}) = f(\vec{x})$ *and the runtime of* $\mathsf{EvalPoly}(\widetilde{f}, \vec{x})$ *is* $\mathsf{poly}(m, d', \log q)$.

Using this theorem we will get a PIR scheme where the online bottleneck (the maximum of bandwidth and computation time) scales as $n^{\frac{1}{S^{1-1/\omega(1)}}}$. Here the scaling of the online bottleneck is minimized.

The main difference of our PIR schemes here compared to that in Section 4.2 is that our schemes use the PolyPreprocess from the above theorems in the preprocessing phase instead of the algorithm in Lemma 4.7. Further, in the online phase, the server uses the data-structure obtained from the preprocessing instead of simply looking up the value. The other difference here is the polynomial F has a restriction on individual degree as opposed to total degree – we need this in order to use the fast polynomial evaluation algorithm. We give the formal description of our construction below.

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

- $\mathbf{Preproc}_s$: Encode database DB to m-variate polynomial F with individual degree at most d'. Note that the total degree d is therefore at most md'. We construct $E : [\![n]\!] \to \mathbb{F}_q^m$ be an *injective* index function, and recover F by interpolating on the set $\{\mathsf{DB}[i]\}_{i \in [\![n]\!]}$ using the techniques described by Lin et al. [LMW23].

  For each $\vec{a} \in A_{<t,m}$, use compute in Lemma 4.7 to compute $\widetilde{\overline{\partial}^{\vec{a}} \circ F} = \mathsf{PolyPreprocess}(\overline{\partial}^{\vec{a}} \circ F)$.
- $\mathbf{Query}$: Given query index i, the client uniformly generates $\vec{v} \in \mathbb{F}_q^m$, and sets $\vec{u} = E(i)$. Then it picks S distinct and **nonzero** elements in $\mathbb{F}_q$ called $\lambda_0, \ldots, \lambda_{S-1}$.

  For $s \in [\![S]\!]$, the client sets
  $$\vec{z}_s = \vec{u} + \lambda_s \vec{v}.$$
  The client sends $Q_s = \vec{z}_s$ to each server $s \in [\![S]\!]$.
- $\mathbf{Answer}_s$: The s-th server ($s \in \{0, 1, \ldots, S\}$) parses the message received from the client as a vector $\vec{z}_s$. For each $\vec{a} \in A_{<t,m}$, it sends
  $$\mathsf{ans}_{s,\vec{a}} = \mathsf{EvalPoly}(\widetilde{\overline{\partial}^{\vec{a}} \circ F}, \vec{z}_s)$$
  back to the client.
- $\mathbf{Recons}$:
  1. Define univariate polynomial $f(\lambda) = F(\vec{u} + \lambda\vec{v})$, clearly $\vec{z}_s = \vec{u} + \lambda_s\vec{v} = f(\lambda_s)$. Given the responses of all servers, the client computes $\overline{\partial}^{(k)} f(\lambda_s)$ for all $s \in [\![S]\!]$ and $0 \leq k < t$ by:
  $$\begin{aligned}\overline{\partial}^{(k)} f(\lambda_s) &= \sum_{\vec{a} \in A_{k,m}} \overline{\partial}^{\vec{a}} \circ F(\vec{z}_s) \cdot \vec{v}^{\vec{a}} \\ &= \sum_{\vec{a} \in A_{k,m}} \mathsf{ans}_{s,\vec{a}} \cdot \vec{v}^{\vec{a}}\end{aligned}$$

2. Reconstruct f by its Hasse derivatives. Since F has degree d, f has degree at most d. For each $s \in [\![S]\!]$, the client has already known its k-th order Hasse derivatives at point λ_s for any $0 \leq k < t$. So, if $S \cdot t > d$, the client can use Hermite interpolation (Lemma 4.4) to reconstruct the coefficients, and output $f(0) = F(\vec{u}) = \mathsf{DB}[i]$ as the answer.

The correctness follows from the scheme in Section 4.2 and the correctness of the fast polynomial evaluation algorithm. Security follows from the scheme in Section 4.2.

In terms of efficiency, the bandwidth and client computation remain the same as in Section 4.2. The online server computation, server storage and the preprocessing computation depend on the particular polynomial preprocessing scheme used.

5.1 Minimizing Bandwidth Subject to Scalability

Instantiating our scheme with the fast polynomial evaluation algorithm from [BGG+24], we get the following theorem that would help us obtain a scalable scheme where the bandwidth is minimized.

Theorem 5.3. *Let $n, S, d', m, q, t \in \mathbb{N}$ satisfy the following constraints.*

- *q is a prime or a prime power with $q \geq S + 1$,*
- *$q^m \geq n$,*
- *$\binom{m+d' \cdot m}{m} \geq n$,*
- *$S \cdot t > d' \cdot m$.*

Then, there exists an S-server PIR scheme which achieves the following parameters.

- *per-server upload bandwidth $m \log q$, per-server download bandwidth $\binom{m+t-1}{m} \log q$.*
- *per-server computation $O\left(\binom{m+t-1}{m} \cdot 16^m \cdot \mathsf{poly}(m, d', \log q)\right)$.*
- *client computation $O\left(S \cdot \binom{m+t-1}{m} \cdot m \cdot \mathsf{poly} \log q\right) + O(\mathsf{poly}(d', m, \log q))$.*
- *per-server storage $O\left(\binom{m+t-1}{m} \cdot (16d'(\log d' + \log\log q))^m \cdot m \cdot \mathsf{poly}(m, d', \log q)\right)$.*
- *per-server preprocessing time $O\left(\binom{m+t-1}{m} \cdot (16d'(\log d' + \log\log q))^m \cdot m \cdot \mathsf{poly}(m, d', \log q)\right)$.*

From this theorem, we get a preprocessing PIR scheme where with $\omega(1)$ servers, the server storage is $n^{1+o(1)}$ and the bandwidth scales as $n^{\widetilde{O}(1/S)}$. Concretely, we set $m = \lceil \log n / \log\log S \rceil$, $d' = \lceil n^{1/m} \rceil$, $t = \lceil (m \cdot d' + 1)/S \rceil$ and q to be the smallest integer greater or equal to $S + 1$ that is a prime. From Bertrand's postulate, $q \leq 2S$. It is easy to verify that all the constraints are satisfied with this parameter choice. Further, notice that $d' \leq (n^{1/m} + 1) \leq 1 + \log S$ since $1/m \leq \log\log S / \log n$. We also have that $m/t \leq S/d' < S/\log S$. Also, observe that

$$\begin{aligned} t &\leq (m \cdot d' + 1)/S + 1 \leq ((\log n / \log\log S + 1) \cdot (\log S + 1) + 1)/S + 1 \\ &= \log n \frac{\log S + 1}{S \log\log S} + \frac{\log S}{S} + \frac{1}{S} + 1 \leq \log n \frac{\log S + 1}{S \log\log S} + 2 \,. \end{aligned}$$

The last inequality follows since $(1+\log S)/S \leq 1$ for any integer $S \geq 2$. We can now calculate an upper bound on $\binom{m+t-1}{t-1}$ using the above observations.

$$\begin{aligned}\binom{m+t-1}{t-1} \leq \binom{m+t}{t} &\leq (e(m+t)/t)^t \leq (2em/t)^t \\ &\leq (2eS/\log S)^{\log n \frac{\log S+1}{S \log\log S}+2} \\ &= (2eS/\log S)^2 \cdot n^{\frac{(1+\log e+\log S-\log\log S)(\log S+1)}{S\log\log S}} \\ &\leq n^{O(\log^2 S/S\log\log S)+O(\log S/\log n)} .\end{aligned}$$

The last inequality above uses $\log e \leq 2$. Moreover, $16^m \leq 16^{\log n/\log\log S+1} = n^{4/\log\log S} \cdot 16$. Further,

$$\begin{aligned}(d')^m &\leq (1+\log S)^{\log n/\log\log S+1} = (1+\log S) \cdot n^{\log(1+\log S)/\log\log S} \\ &\leq (1+\log S) \cdot n^{(1+\log\log S)/\log\log S} \\ &\leq n^{1+1/\log\log S+O(\log S/\log n)} .\end{aligned}$$

The third step uses the fact $\log(1+\log S) \leq \log(2\log S) = 1+\log\log S$ since $S \geq 2$. We also have that

$$\begin{aligned}(\log d' + \log\log q)^m &\leq (\log(1+\log S)+\log\log 2S)^{\log n/\log\log S+1} \\ &\leq (2\log\log 2S) \cdot n^{(1+\log\log\log 2S)/\log\log S} \\ &\leq n^{O(\log\log\log 2S/\log\log S)+O(\log\log\log S/\log n)}\end{aligned}$$

We then get the following corollary.

Corollary 5.4. *There is an S-server PIR scheme which achieves the following parameters.*

- *per-server bandwidth* $n^{O\left(\frac{\log^2 S}{S\log\log S}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n,S)$,
- *per-server computation* $n^{O\left(\frac{\log^2 S}{S\log\log S}\right)+O\left(\frac{1}{\log\log S}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n,S)$,
- *client computation* $n^{O\left(\frac{\log^2 S}{S\log\log S}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n,S)$,
- *per-server preprocessing time and storage*

$$n^{1+O\left(\frac{\log^2 S}{S\log\log S}\right)+O\left(\frac{\log\log\log S}{\log\log S}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n,S) .$$

For $S = \omega(1)$, this scheme has storage and preprocessing time $n^{1+o(1)}$ and online computation and bandwidth $n^{o(1)}$. Moreover, the bandwidth scales roughly $n^{\widetilde{O}(1/S)}$. Here the server computation and storage are also scalable, but their scaling is worse than the bandwidth.

Remark 5.5. We note that we if we used the fast polynomial evaluation algorithm from [BGKM22] directly (which is used by [LLFP24]), we would have not achieved a scheme that has similar scalability and leads to a doubly efficient construction with a super-constant number of servers. The preprocessing time and space for that algorithm is $O(4d' \cdot \alpha \cdot p)^m \cdot \mathsf{poly}\log(m,d',p)$ where $q = p^\alpha$. It is easy to see that either p or α is greater than $\log q/\log\log q$.

Further, we note the following points for a doubly efficient construction.

- Since $(d')^m$ has to be $O(n)$ as required by the constraints, the polynomial preprocessing will only lead to an advantage as opposed to simply storing all evaluations (as we do in our construction in Section 4.2) when $q^m \gg n$, i.e., $m \gg \log n / \log q$. So, to get any significant advantage from using the polynomial preprocessing, we would need m to be at least $\log n \cdot g(n) / \log q$ where $g(n) = \omega(1)$.
- Moreover since $d' = p^\alpha$, and we set $(d')^m = O(n)$, it follows that $(p\alpha)^m = O(n^{1/\alpha} \cdot \alpha^m)$. Since $\alpha = \omega(1)$ (as they need the condition that $p = d^{o(1)}$), $n^{1/\alpha} = n^{o(1)}$. We also need α^m to be $n^{o(1)}$ for the construction to be doubly efficient. We have that $(\alpha^m) = n^{\alpha \cdot g(n)/\log q}$. Since we set $q = S + 1$, this means that for $\alpha^m = n^{o(1)}$ we need that $\log q = \omega(\alpha \cdot g(n))$, i.e., $S = 2^{\omega(1)}$. So, we would need many more servers to get a doubly efficient construction.

This shows that we necessarily need the new polynomial preprocessing algorithm from [BGG+24] to achieve our result.

Remark 5.6 (Comparison with [LLFP24])**.** Setting $S = O(\log n \log\log n / \log\log\log n)$ in Corollary 5.4, recovers the first construction in [LLFP24] with the only difference being the difference in the preprocessing algorithm used. The asymptotic parameters are same.

Setting $m = O(\log n/(\log\log n - \log\log\log n)), d' = O(\log n/\log\log n), S = O(\log n/\log\log n), q = S + 1, t = m + 1$ in Theorem 5.3, we get a scheme that is strictly better than construction 2 in [LLFP24] in terms of number of servers, preprocessing time and space, server computation, client computation and bandwidth.

5.2 Minimizing Online Bottleneck Cost Subject to Scalability

Instantiating this scheme with the fast polynomial evaluation scheme of [KU08], gives us this theorem.

Theorem 5.7. *Let $n, S, d', m, q, t \in \mathbb{N}$ satisfy the following constraints.*

- *q is a prime or a prime power with $q \geq S + 1$,*
- *$q^m \geq n$,*
- *$\binom{m+d' \cdot m}{m} \geq n$,*
- *$S \cdot t > d' \cdot m$.*

Then, there exists an S-server PIR scheme which achieves the following parameters.

- *per-server upload bandwidth $m \log q$ and download bandwidth $\binom{m+t-1}{m} \log q$.*
- *per-server computation $O\left(\binom{m+t-1}{m} \cdot \mathsf{poly}(m, d', \log q)\right)$.*
- *client computation $O\left(S \cdot \binom{m+t-1}{m} \cdot m \cdot \mathsf{poly} \log q\right) + O(\mathsf{poly}(d', m, \log q))$.*
- *per-server storage $O\left(\binom{m+t-1}{m} \cdot (md'(\log d' + \log m + \log\log q))^m \cdot m \cdot \mathsf{poly}(m, d', \log q)\right)$.*
- *per-server preprocessing time $O\left(\binom{m+t-1}{m} \cdot (md'(\log d' + \log m + \log\log q))^m \cdot m \cdot \mathsf{poly}(m, d', \log q)\right)$.*

We next set parameters for this scheme. We need to have $m \geq \log n / \log q$ and $q \geq S + 1$ because of the constraints. We abuse notation and let $\omega(1)$ be any arbitrary superconstant function such that it is $o(\log n)$ (we assume this because it leads to simplified expressions.). So, we set $m = \lceil \log n \cdot \omega(1) / \log S \rceil$. To satisfy the constraints, while minimizing the PIR parameters we set $d' = \lceil S^{1/\omega(1)} \rceil$, q to be the smallest

prime greater or equal to $S+1$ (From Bertrand's postulate, $q \leq 2S$), $t = \lceil (md'+1)/S \rceil$. We observe the following.

$$\begin{aligned}
(d')^m &\leq (S^{1/\omega(1)}+1)^{\log n \cdot \omega(1)/\log S + 1} \\
&= (S^{1/\omega(1)}+1) \cdot n^{\log(S^{1/\omega(1)}+1)/\log(S^{1/\omega(1)})} \\
&= (S^{1/\omega(1)}+1) \cdot n^{1+\log(1+1/S^{1/\omega(1)})/\log(S^{1/\omega(1)})} \\
&\leq (S^{1/\omega(1)}+1) \cdot n^{1+\log e/(S^{1/\omega(1)}\log(S^{1/\omega(1)}))} \\
&\leq (S^{1/\omega(1)}+1) \cdot n^{1+1.4427/(S^{1/\omega(1)}\log(S^{1/\omega(1)}))} \\
&= n^{1+\widetilde{O}(1/S^{1/\omega(1)})+O(\log S/\log n)}
\end{aligned}$$

We have used the fact that for $a \geq 1$, $\log(1+1/a) \leq (\log e)/a$. We also have that

$$\begin{aligned}
m^m &\leq (\log n \cdot \omega(1)/\log S + 1)^{(\log n \cdot \omega(1)/\log S + 1)} \\
&\leq (\log n \cdot \omega(1)/\log S + 1) \cdot n^{\log(\log n \cdot \omega(1)/\log S + 1)\cdot \omega(1)/\log S} \\
&\leq (\log n \cdot \omega(1)/\log S + 1) \cdot n^{((1+\log(\log n \cdot \omega(1)/\log S))\cdot \omega(1)/\log S} \\
&= (\log n \cdot \omega(1)/\log S + 1) \cdot n^{\frac{(1+(\log\log n + \log\omega(1) - \log\log S)\cdot\omega(1)}{\log S}} \\
&\leq n^{O(\frac{\log\log n \cdot \omega(1)}{\log S})}
\end{aligned}$$

Furthermore,

$$\begin{aligned}
&(\log d' + \log m + \log\log q)^m \\
&\leq \log\left((S^{1/\omega(1)}+1)\cdot\left(\frac{\log n\cdot\omega(1)}{\log S}+1\right)\cdot\log(2S)\right)\cdot n^{\frac{\log(\log(S^{1/\omega(1)}+1)\cdot(\log n\cdot\omega(1)/\log S+1)\cdot\log(2S))\cdot\omega(1)}{\log S}} \\
&\leq O(\log S/\omega(1) + \log\log n)\cdot n^{O(\log S/\omega(1)+\log\log n)\cdot\omega(1)/\log S} \\
&\leq n^{O(\log\log n\cdot\omega(1)/\log S)}
\end{aligned}$$

It is easy to see that $m/t \leq S/d' \leq S^{1-1/\omega(1)}$. This gives us

$$\begin{aligned}
t &\leq md'/S + 1 \leq (\log n\cdot\omega(1)/\log S+1)\cdot(S^{1/\omega(1)}+1)/S + 1 \\
&\leq \log n \frac{\omega(1)\cdot S^{1/\omega(1)}+\omega(1)}{S\log S} + \frac{1}{S} + \frac{1}{S^{1-\omega(1)}} + 1 \\
&\leq \log n \frac{\omega(1)\cdot S^{1/\omega(1)}+\omega(1)}{S\log S} + 3
\end{aligned}$$

Therefore, we have that

$$\begin{aligned}
\binom{m+t-1}{t-1} \leq \binom{m+t}{t} &\leq (e(m+t)/t)^t \leq (2em/t)^t \\
&\leq (2eS^{1-1/\omega(1)})^{\log n\frac{\omega(1)\cdot S^{1/\omega(1)}+\omega(1)}{S\log S}+3} \\
&= (2eS^{1-1/\omega(1)})^3 \cdot n^{\frac{(1+\log e+(1-1/\omega(1))\log S)\cdot\omega(1)}{S^{1-1/\omega(1)}\log S}} \\
&\leq (2eS^{1-1/\omega(1)})^3 \cdot n^{\frac{2.4427\omega(1)+(\omega(1)-1)\log S}{S^{1-1/\omega(1)}\log S}} \\
&= n^{\widetilde{O}(1/S^{1/\omega(1)})+O(\log S/\log n)}
\end{aligned}$$

We get the following corollary.

Corollary 5.8. *Let $\omega(1)$ be any superconstant function such that it is in $o(\log n)$. There is an S-server PIR scheme which achieves the following parameters.*

- *per-server bandwidth* $n^{\widetilde{O}\left(\frac{1}{S^{1/\omega(1)}}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n) \cdot \log S$,
- *per-server computation* $n^{\widetilde{O}\left(\frac{1}{S^{1/\omega(1)}}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n, S)$,
- *client computation* $n^{\widetilde{O}\left(\frac{1}{S^{1/\omega(1)}}\right)+O\left(\frac{\log S}{\log n}\right)} \cdot \mathsf{poly}\log(n, S)$,
- *per-server preprocessing time and storage*

$$n^{1+\widetilde{O}\left(\frac{1}{S^{1/\omega(1)}}\right)+O\left(\frac{\log S}{\log n}\right)+O\left(\frac{\log\log n\cdot\omega(1)}{\log S}\right)} \cdot \mathsf{poly}\log(n, S) \ .$$

Remark 5.9. Observe that when S is superpolylogarithmic in n and $S \in n^{o(1)}$, the storage becomes $n^{1+o(1)}$. The bandwidth, computation parameters also become $n^{o(1)}$. Therefore, the construction is doubly efficient when S is superpolylogarithmic in n. Note that this $S \in n^{o(1)}$ is *not an additional restriction* because for $S \gg n^{o(1)}$, the total client computation and bandwidth is naturally $\gg n^{o(1)}$ and the construction cannot be doubly efficient.

6 Multi-Server PIR for the Polynomial Space Setting

In this section, we want to minimize the bandwidth but still keep server storage polynomial in the size of the database. To do so, we add some optimizations to our construction in Section 4.2. We obtain an S-server PIR that is a strict improvement Beimel et al. [BIM04]. Specifically, while both our new scheme and Beimel et al. [BIM04] achieve $n^{(1+\epsilon)/S}$ bandwidth and computation, our scheme achieves a polynomial improvement in the server space and preprocessing cost for every $S \geq 3$. Then, we apply the generic balancing trick of Section 7 to further reduce the bandwidth to $n^{(1+\epsilon)/(S+1)}$.

PIR scheme with additional optimizations. We first describe a base S-server PIR scheme that achieves $O(n^{(1+\epsilon)/S})$ bandwidth. This scheme has exactly same bandwidth and computation as [BIM04, Theorem 4.5], but the preprocessing time and server storage have been further improved by a factor of $S/\log S$ over the exponent. This improvement stems from our additional optimizations. Like the construction in Section 4.2, this scheme uses multiplicity codes as well. However, we encode the database differently, and use a different reconstruction technique.

Concretely, we let $E : [\![n]\!] \to \{0,1\}^m$ be an injective function that maps the indices of a n-bit DB to bit-strings of length m and hamming weight *exactly* d. We will choose $m = O(\log n)$ and $d = \theta m$ for some $0 < \theta \leq 1/2$ – the following lemma shows that the choice of such d is possible.

Lemma 6.1. *For any constant $0 < \theta \leq 1/2$, if we want $\binom{m}{\theta m} \geq n$ to hold, for sufficiently large n, it suffices to set $m = \frac{\log n}{H(\theta)}(1 + o(1))$, where $H(p) = -p\log_2 p - (1-p)\log_2(1-p)$ is the binary entropy function, and $o(1)$ hides a function that goes to 0 as n goes to infinity.*

We defer the proof of this lemma to Appendix D.

We let $\mathbb{F}_q$ be a finite field such that $S \leq q \leq 2S - 1$ (Bertrand's postulate guarantees existence of such q). Note that here we can potentially choose $q = S$ – this reduction in field size by 1 results in saving polynomial factors in server storage. We define a polynomial $F \in F_q[X_1, \ldots, X_m]$ as

$$F(\vec{X}) = \sum_{i \in [\![n]\!]} \mathsf{DB}[i] \cdot \vec{X}^{E(i)} \ . \tag{4}$$

Note that because the range of $E(j)$ is $\{0,1\}^m$ and each $E(j)$ has hamming weight d, F is a multilinear polynomial of degree d. Further, even in this case $F(E(i)) = \mathsf{DB}[i]$ for $i \in [\![n]\!]$. However, we will use a different reconstruction procedure here.

Suppose the client wants to query the database at index i such that $E(i) = \vec{u}$. The client chooses a random $\vec{v} \in \mathbb{F}_q^m$ and distinct field elements $\lambda_0, \ldots, \lambda_{S-1}$ and sends $\vec{z}_s = \lambda_s \vec{u} + \vec{v}$ to server s for $s \in [\![S]\!]$. Note that this is different from what we did in the construction in Section 4.2. For $s \in [\![S]\!]$, server s sends back $\boldsymbol{\partial}^{\vec{a}} \circ F(\vec{z}_s)$ for $a \in A_{<\lceil (d+1)/S \rceil, m, 1}$ to the client. We define $f(\lambda) = F(\lambda \vec{u} + \vec{v})$. Given the values the client receives from the server, it can reconstruct the polynomial f using Hermite interpolation. It turns out that $\mathsf{DB}[i]$ is the co-efficient of λ^d in this polynomial. We prove this in Claim 6.3.

Remark 6.2. Essentially, we encode a database with a multiplicity code of order-$\lceil (d+1)/S \rceil$ evaluations of degree-d multilinear polynomials in m variables, and use the multilinear property to give an alternate reconstruction technique in contrast to the technique in Section 4.2, which is the usual idea used to decode multiplicity codes.

In order to make the server's online computation efficient, we make the server store $\partial^{\vec{a}} \circ F(\vec{x})$ for all $\vec{a} \in A_{<\lceil (d+1)/S \rceil, m, 1}$, $\vec{x} \in \mathbb{F}_q^m$. We do this efficient by applying the lemma 4.7 to the polynomial $\partial^{\vec{a}} \circ F$.

We now describe our construction formally.

6.0.1 Construction

Parameters and notations. We will choose the following parameters.

- Let $\mathbb{F}_q$ be a finite field with order $q \geq S$, by Bertrand's postulate, q is bounded by $2S - 1$.
- 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 $0 < \theta \leq 1/2$, such that $\binom{m}{d} \geq n$ — this is possible due to Lemma 6.1. We will choose $t = \lceil (d+1)/S \rceil$, so we have that $t \cdot S > d$.
- We use the following polynomial F over $\mathbb{F}_q$ to encode database.

$$F(\vec{X}) = \sum_{i \in [\![n]\!]} \mathsf{DB}[i] \cdot \vec{X}^{E(i)}$$

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

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

- $\mathbf{Preproc}_s$: The same holds for each server $s \in [\![S]\!]$: for each $\vec{a} \in A_{<t,m,1}$, each $\vec{x} \in \mathbb{F}_q^m$, calculate $\boldsymbol{\partial}^{\vec{a}} \circ F(\vec{x})$, and store all results, this step can be efficiently implemented by applying Lemma 4.7 to polynomial $\boldsymbol{\partial}^{\vec{a}} \circ F$.
- $\mathbf{Query}$: Let $i \in [\![n]\!]$ be the queried index. Let vector $\vec{u} = E(i) \in \mathbb{F}_q^m$. The client first picks S distinct elements in $\mathbb{F}_q$ called $\lambda_0, \ldots, \lambda_{S-1}$, then randomly picks $\vec{v} \in \mathbb{F}_q^m$.

  For each server s, the client sets

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

  The client then sends $Q_s = \vec{z}_s$ to each server $s \in [\![S]\!]$.

- **Answer**$_s$: The s-th server parses the message received from the client as vector $\vec{z}_s$. For each $\vec{a} \in A_{<t,m,1}$, it sends back

$$\mathsf{ans}_{s,\vec{a}} = \underbrace{\boldsymbol{\partial}^{\vec{a}} \circ F(\vec{z}_s)}_{\text{precomputed during preproc}}$$

- **Recons**:

1. Define univariate polynomial $f(\lambda) = F(\lambda\vec{u}+\vec{v})$, clearly $\vec{z}_s = \vec{v}+\lambda\vec{u} = f(\lambda_s)$. The client computes the Hasse derivatives $\overline{\partial}^{(k)} f(\lambda_s)$ for all $s \in [\![S]\!]$ and $0 \le k < t$ by the chain rule (Lemma 4.3):

$$\overline{\partial}^{(k)} f(\lambda_s) = \sum_{\vec{a}\in A_{k,m,1}} \boldsymbol{\partial}^{\vec{a}} \circ F(\vec{z}_s) \cdot \vec{u}^{\vec{a}} = \sum_{\vec{a}\in A_{k,m,1}} \mathsf{ans}_{s,\vec{a}} \cdot \vec{u}^{\vec{a}} \tag{5}$$

2. Reconstruct f by its Hasse derivatives. It is easy to see that f has degree at most d. For each $s \in [\![S]\!]$, the client has already known its k-th order Hasse derivatives at point λ_s for any $0 \le k < t$. Since $S \cdot t > d$, the coefficients of f can be recovered by Hermite interpolation (Lemma 4.4).
Finally, the client outputs the highest term of f, i.e., $\mathsf{Coeff}_{\lambda^d}(f(\lambda))$ as the answer.

6.0.2 Proof of Correctness

Correctness follows from the following claim.

Claim 6.3. Let $t, d, m > 0$ and $S > 1$ be integers. Let q be a prime or prime power. Let $F \in F_q[X_1, \ldots, X_m]$ be the polynomial defined in Equation (4). Let $i \in [\![n]\!]$ and $\vec{u} = E(i)$ where E is the encoding defined above. Let $\vec{v} \in \mathbb{F}_q^m$. Let $\vec{z}_s = \lambda_s\vec{u} + \vec{v}$ for $s \in [\![S]\!]$ for distinct $\lambda_0, \lambda_1, \ldots, \lambda_{S-1} \in \mathbb{F}_q$. Then, given the evaluation of $\boldsymbol{\partial}^{\vec{a}} \circ F(\vec{z}_s)$ for all $\vec{a} \in A_{<t,m,1}$, $s \in S$, we can recover $\mathsf{DB}[i]$ if $S \cdot t > d$.

Proof. Let $f(\lambda) = F(\lambda\vec{u} + \vec{v})$. We have that for $k \in [\![t]\!]$

$$\overline{\partial}^{(k)} f(\lambda_s) = \sum_{\vec{a}\in A_{k,m,1}} \overline{\partial}^{\vec{a}} \circ F(\vec{z}_s) \cdot \vec{u}^{\vec{a}} = \sum_{\vec{a}\in A_{k,m,1}} \boldsymbol{\partial}^{\vec{a}} \circ F(\vec{z}_s) \cdot \vec{u}^{\vec{a}} \,.$$

The second equality follows from Lemma 4.3 because F is multilinear. Now if $t \cdot S > d$, we can reconstruct f using Lemma 4.4.

If we can reconstruct f, we claim that $\mathsf{DB}[i]$ is the coefficient of λ^d in f. To see this, first observe that $\vec{u}$ has hamming weight exactly d from the definition of E. So, for any $\vec{w} \in \{0,1\}^d$ such that $\vec{w}$ has hamming weight d, we have

$$\mathsf{Coeff}_{\lambda^d}\left((\lambda\vec{u} + \vec{v})^{\vec{w}}\right) = \mathsf{Coeff}_{\lambda^d}\left(\prod_{\ell=1}^{m}(\lambda \cdot u_\ell + v_\ell)^{w_\ell}\right) = \begin{cases} 1 \text{ if } \vec{w} = \vec{u} \\ 0 \text{ otherwise} \end{cases} .$$

The second equality above follows because the coefficient of λ^d is non-zero if and only if there are d or more ℓ's such that $u_\ell = 1$ and $w_\ell = 1$. Now if $\vec{u} \neq \vec{w}$, there are less than d such ℓ's because the hamming weights of $\vec{u}, \vec{w}$ are d. When $\vec{u} = \vec{w}$, the coefficient of λ^d is 1 because $u_\ell = 1$ for all $\ell \in [m]$.

Therefore, it follows that

$$\begin{aligned}\mathsf{Coeff}_{\lambda^d}(f(\lambda)) &= \mathsf{Coeff}_{\lambda^d}\left(\sum_{j\in[\![n]\!]} \mathsf{DB}[j] \cdot (\lambda\vec{u} + \vec{v})^{E(j)}\right) \\ &= \sum_{j\in[\![n]\!]} \mathsf{DB}[j] \cdot \mathsf{Coeff}_{\lambda^d}\left((\lambda\vec{u} + \vec{v})^{E(j)}\right) = \mathsf{DB}[i] \,.\end{aligned}$$

The second equality above follows because E is an injective mapping and therefore $E(j) = \vec{u} \implies j = i$. This concludes the proof. □ □

6.0.3 Proof of Security

The privacy proof is easy to see: for each server s, since $\vec{v} \xleftarrow{\$} \mathbb{F}_q^m$ is randomly sampled, $\vec{z}_s = \vec{v} + \lambda_s \vec{u}$ is also randomly distributed in $\mathbb{F}_q^m$, so the message received by s-th server doesn't reveal any nontrivial information.

6.0.4 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, the client will send a vector $\vec{z}_s \in \mathbb{F}_q^m$ to the server. Recall that $m = \frac{\log n}{H(\theta)}(1+o(1))$ (Lemma 6.1) and each element in $\mathbb{F}_q$ takes $O(\log S)$ space since $q < 2S$, so the per-server upload bandwidth is
$$O(m \log S) = n^{o(1)} \log S.$$
For each server s and each $\vec{a} \in A_{<t,m,1}$, the server returns answer $\mathsf{ans}_{s,\vec{a}} \in \mathbb{F}_q$. By the fact that $|A_{<\lceil (d+1)/S \rceil, m, 1}| = \Lambda(m, \lceil (\theta m - S + 1)/S \rceil) \le 2^{H(\theta/S)m}$ for $0 < \theta \le 1/2$, the per-server download bandwidth is
$$O(|A_{<\lceil (d+1)/S \rceil, m, 1}| \log S) = n^{((1+o(1))H(\theta/S)/H(\theta)} \log S.$$
- **Server computation**: For each server s and each $\vec{a} \in A_{<\lceil (d+1)/S \rceil, m, 1}$, the server only needs send back one element $\mathsf{ans}_{s,\vec{a}}$ and takes time $O(\log S)$, so the computation of each server is bounded by
$$O(|A_{<\lceil (d+1)/S \rceil, m, 1}| \log S) = n^{(1+o(1))H(\theta/S)/H(\theta)} \log S.$$
- **Client computation**: First the client computation is not less than the bandwidth i.e. $O(mS \log S + |A_{<\lceil (d+1)/S \rceil, m, 1}| S \log S)$. Then we consider the time complexity of **Recons**.
Since the Hermite interpolation takes only $\mathsf{poly}(d, \log q) = \mathsf{poly}(\log n, \log S)$ time (Lemma 4.4), the time complexity of **Recons** is bounded by the arithmetic operations in $\mathbb{F}_q$ to reconstruct the Hasse derivatives of g (see Equation (5)).
For each server s and $\vec{a} \in A_{<\lceil (d+1)/S \rceil, m, 1}$, the client should do $O(m) = n^{o(1)}$ multiplications in $\mathbb{F}_q$ (where each takes $O(\log^2 S)$ time), since there are S servers, the total client computation is bounded by
$$\begin{aligned} &O(mS \log S + |A_{<\lceil (d+1)/S \rceil, m, 1}| \cdot Sm \log^2 S) \\ =&n^{(1+o(1))H(\theta/S)/H(\theta)} S \log^2 S. \end{aligned}$$
- **Server space**: Recall that we use a precompute-all approach: for each $\vec{a} \in A_{<\lceil (d+1)/S \rceil, m, 1}$ and each $\vec{x} \in \mathbb{F}_q^m$, each server stores an element in $\mathbb{F}_q$. The server space is
$$\begin{aligned} O(|A_{<\lceil (d+1)/S \rceil, m, 1}| q^m \log S) &= q^m n^{(1+o(1))H(\theta/S)/H(\theta)} \log S \\ &= n^{(1+o(1))(\log q + H(\theta/S))/H(\theta)} \\ &\le n^{(1+o(1))(\log S + 1 + H(\theta/S))/H(\theta)} \end{aligned}$$
- **Preprocessing time**: Each element stored by each server can be computed in an amortized $\mathsf{poly}(m, \log q) = \mathsf{poly}(\log n, \log S)$ time (Lemma 4.7), so the preprocessing time is bounded by $n^{(1+o(1))(\log q + H(\theta/S))/H(\theta)} \le n^{(1+o(1))(\log S + 1 + H(\theta/S))/H(\theta)}$.

This gives us the following theorem.

Theorem 6.4 (Base scheme in the polynomial storage setting)**.** *For any $\epsilon \in (0,1)$ and $0 < \theta \leq 1/2$, there exists an S-server PIR scheme which achieves $n^{(1+o(1))\cdot H(\theta/S)/H(\theta)} \log S$ per-server bandwidth, $n^{(1+o(1))\cdot H(\theta/S)/H(\theta)} \log S$ per-server computation and $n^{(1+o(1))\cdot H(\theta/S)/H(\theta)} S \log^2 S$ client computation per query, with $n^{(1+o(1))(\log q + H(\theta/S))/H(\theta)}$ preprocessing time and server storage where $\mathbb{F}_q$ is the minimum field such that $q \geq S$. Specifically, when S is a constant, the preprocessing time and server storage are bounded by* $\mathsf{poly}(n)$.

Analyzing the concrete $\mathsf{poly}(n)$ **for large** 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 the constant θ to be sufficiently small, we can achieve $n^{(1+\epsilon)/S}$ bandwidth and computation per query for any constant $\epsilon > 0$. Further, since S and θ are both constants, the server space and preprocessing time is bounded by some polynomial in n.

We can further characterize the server space and preprocessing cost. For sufficiently small θ,

$$\frac{H(\theta/S)}{H(\theta)} - 1/S \leq \frac{\ln(S)}{S(1+\ln(\frac{1}{\theta}))} \cdot (1 + O(\theta)) \tag{6}$$

If we want to achieve $n^{(1+o(1))(1+\epsilon)/S} \log S$ bandwidth and server computation and $n^{(1+o(1))(1+\epsilon)/S} S \log^2 S$ client computation, we can choose Equation (6) to be ϵ/S, i.e.,

$$\frac{\ln S}{S(1+\ln(\frac{1}{\theta}))} \cdot (1 + O(\theta)) = \epsilon/S$$

Thus it suffices to set $1 + \ln(1/\theta) = (1 + o_1(1))\frac{\ln S}{\epsilon}$ for some function $o_1(1)$ that goes to 0 as ϵ goes to 0. Therefore, $\theta = 1/\exp((1 + o_1(1)) \ln S/\epsilon - 1)$. In this case, the server space and preprocessing cost is upper bounded by

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

In particular,

$$(\log S + 1)/H(\theta) \leq \frac{\ln 2(\log S + 1)}{\theta(1 + \ln \frac{1}{\theta})}(1 + o_2(1))$$

where $o_2(1)$ is a function on θ that goes to 0 as θ goes to 0. Therefore, we have

$$\begin{aligned}
(\log S + 1)/H(\theta) &\leq \frac{\ln 2(\log S + 1) \cdot \exp\left(\frac{(1+o_1(1)) \ln S}{\epsilon} - 1\right)}{(1 + o_1(1))\frac{\ln S}{\epsilon}} \cdot (1 + o_2(1)) \\
&\leq \frac{\ln 2(\log S + 1) \cdot \epsilon \cdot S^{(1+o_1(1))/\epsilon}}{\ln S \cdot e} \cdot (1 + o_2(1)) \\
&\leq \frac{1}{e} \cdot \frac{\log S + 1}{\log S} \cdot S^{(1+o_1(1))/\epsilon} \cdot (1 + o_2(1)) \\
&\leq 0.3678(1 + o(1)) \cdot \epsilon S^{\frac{1+o(1)}{\epsilon}}
\end{aligned}$$

where $o(1)$ hides terms that go to 0 as S goes to infinity or as ϵ goes to 0. Therefore, for sufficiently small ϵ, sufficiently large S and n, the server space and preprocessing cost is upper bounded by $n^{0.3679\epsilon S^{(1+o(1))/\epsilon}}$.

In summary, for sufficiently small ϵ, sufficiently large S and n, we can achieve $n^{(1+\epsilon)/S} \log S$ bandwidth and server computation, $n^{(1+\epsilon)/S} S \log^2 S$ client computation, and $n^{0.368\epsilon S^{(1+o(1))/\epsilon}}$ server space and preprocessing cost where the $o(1)$ term is a function that goes to 0 as ϵ goes to 0, S and n go to infinity. To get the above, observe that for sufficiently large S and n and sufficiently small ϵ, we can use $1 + \epsilon$ to absorb

$(1+o(1))(1+\epsilon')$ for some $\epsilon > \epsilon'$, and we can use 0.368 to absorb $0.3679 \cdot (1+o(1))$. Further, the $o(1)$ term in the exponent of $S^{(1+o(1))/\epsilon}$ becomes a little larger than before when we substitute the ϵ' with ϵ. With more careful analysis and using the proof in Appendix D, the $o(1)$ in the exponent of $S^{(1+o(1))/\epsilon}$ actually hides $o(\epsilon) + O(\log\log n/\log n)$ terms when we use $1+\epsilon$ to absorb $(1+o(1))(1+\epsilon')$.

Corollary 6.5 (Our base scheme: for large S)**.** *For sufficiently large S, n, and sufficiently small $\epsilon > 0$, there exists an S-server PIR scheme where each query incurs $O(n^{(1+\epsilon)/S}\log S)$ per-server bandwidth and computation, and $O(n^{(1+\epsilon)/S} S\log^2 S)$ client computation, while requiring $n^{0.368\epsilon S^{(1+o(1))/\epsilon}}$ per-server preprocessing time and space.*

Remark 6.6. It is not hard to see that the above Corollary 6.5 also holds any prime or prime power S, as long as n is sufficiently large and ϵ is sufficiently small. This is because for a prime or prime power S, we choose $q = S$, so the $\log S + 1$ term can be replaced with $\log S$, and we need not rely on the "sufficiently large S" condition to absorb the $+1$ term into the $o(1)$ part. In fact, the expression $\log(q)/\log(S)$ is maximized when $S = 6$ and $q = 7$. In this case, the server space and preprocessing cost is upper bounded by $n^{0.4\epsilon S^{(1+o(1))/\epsilon}}$ for sufficiently large n and sufficiently small ϵ. Therefore, the same bound $n^{0.4\epsilon S^{(1+o(1))/\epsilon}}$ also holds for any S as long as n is sufficiently large and ϵ is sufficiently small.

Comparison with Beimel et al. We now compare with the scheme of Beimel et al. [BIM04]. For bandwidth and server computation, both schemes achieve $n^{(1+o(1))H(\theta/S)/H(\theta)}\log S$ cost. The server space and preprocessing cost of Beimel et al. [BIM04] is $n^{(1+o(1))(S-1+H(\theta/S))/H(\theta)}$, and ours is $n^{(1+o(1))(\log q+H(\theta/S))/H(\theta)}$ where $\mathbb{F}_q$ is the smallest filed that size is at least S (in other words, q is the minimal prime power that is at least S). For $S = 2$, our scheme chooses $q = 2$ and $\log q = S - 1$. Therefore, for $S = 2$ servers, both schemes achieve the same server space and preprocessing cost. Our server space and preprocessing cost starts to outperform Beimel et al. when $S = 3$ and larger. For $S = 3$, our field size $q = 3$, and $\log q < S - 1$. Specifically, for $S = 3$, Beimel et al.'s constant in the exponent is $(3-1)/\log(3) \approx 1.26$ times larger than ours. For sufficiently large S, n, and sufficiently small ϵ, our server space and preprocessing cost is $n^{0.368\epsilon S^{(1+o(1))/\epsilon}}$ and Beimel et al. [BIM04] has $n^{0.368(S/\log S)\cdot\epsilon S^{(1+o(1))/\epsilon}}$ server space and preprocessing cost — assuming we fix the bandwidth and computation to $n^{(1+\epsilon)/S}$. In other words, their constant in the exponent is a factor of $S/\log S$ larger than our scheme. Table 2 compares the exact exponents of server storage for some concrete server numbers when $\epsilon = 0.5$.

Table 2: **Numerical Experiments for** $\epsilon = 0.5$. The last four columns represent the exponents of communication/work, server storage of our base scheme, server storage of our base scheme without the $\lambda\vec{u} + \vec{v}$ optimization, and server storage of [BIM04, Theorem 4.3], respectively (e.g. 0.75 means $n^{0.75+o(1)}$).

S	q	θ	**Comm./Work**	**Our Storage**		**[BIM04]'s Storage**
				Optimized	**Unoptimized**	
2	2	0.4110	0.75	1.7735	2.3723	1.7735
3	3	0.2259	0.5	2.5563	3.0947	3.0947
4	4	0.1410	0.375	3.7832	4.3318	5.4874
5	5	0.0956	0.3	5.4051	6.4724	9.0947
6	7	0.0687	0.25	8.0230	8.0230	14.0940
7	7	0.0516	0.2143	9.7838	10.4405	20.0667
8	8	0.0402	0.1875	12.5322	13.2314	28.9918
9	9	0.0321	0.1667	15.6510	17.0652	39.2448
10	11	0.0262	0.15	19.9255	19.9255	51.5976

6.1 Applying the Balancing Technique to Reduce Bandwidth

Combining our base S-server PIR scheme with balancing technique (Lemma 7.2), we immediately obtain the desired result:

Corollary 6.7. *For any $\epsilon \in (0,1)$, $0 < \mu \leq 1$ and $0 < \theta \leq 1/2$, there exists an S-server PIR scheme which achieves $(n^{1-\mu+o(1)} + n^{\mu(H(\theta/S)/H(\theta)+o(1))})\log S$ per-server bandwidth, $n^{1-\mu+\mu(H(\theta/S)/H(\theta)+o(1))}\log S$ per-server computation and $(n^{1-\mu+o(1)} + n^{\mu(H(\theta/S)/H(\theta)+o(1))})S\log^2 S$ client computation per query, with $n^{1-\mu+\mu((\log q+H(\theta/S))/H(\theta)+o(1))}$ preprocessing time and server storage where $\mathbb{F}_q$ is the minimum field such that $q \geq S$. Specifically, when S is a constant, the preprocessing time and server storage are bounded by $\mathsf{poly}(n)$.*

For any $1/(S+1) \leq \alpha \leq 1/S$, we may 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,
- $O(n^{S\alpha(H(\theta/S)/H(\theta)+o(1))}S\log^2 S)$ client computation.

Moreover, similar as the previous analysis, for sufficiently large S, n and sufficiently small $\epsilon > 0$, it suffices to choose $\theta = 1/\exp((1+o(1))\ln S/\epsilon - 1)$ for $H(\theta/S)/H(\theta) < (1+\epsilon)/S$ to hold, therefore we have the following corollary:

Corollary 6.8. *For sufficiently large S, n, and sufficiently small $\epsilon > 0$, for any $\alpha \in [1/(S+1), 1/S]$, there exists an S-server PIR scheme such that, which achieves $O(n^{\alpha(1+\epsilon)}\log S)$ per-server bandwidth, $O(n^{1-(S-1-\epsilon)\alpha}\log S)$ per-server computation and $O(n^{\alpha(1+\epsilon)}S\log^2 S)$ client computation per query, with $n^{1-S\alpha+\alpha 0.368\epsilon S^{1+(1+o(1))/\epsilon}}$ preprocessing time and server storage.*

In above theorem, if we take constant S and parameter $\alpha = 1/(S+1)$ to minimize total bandwidth, in which the upload and download bandwidth are balanced up to some $1+\epsilon$ factor of exponents, then the PIR scheme achieves $O(n^{(1+\epsilon)/(S+1)}\log S)$ per-server bandwidth, $O(n^{(2+\epsilon)/(S+1)}\log S)$ per-server computation and $O(n^{(1+\epsilon)/(S+1)}S\log^2 S)$ client computation per query with $\mathsf{poly}(n)$ preprocessing time and server storage.

7 A Generic Balancing Method

In this section, we describe our generic balancing technique. We explained the intuition in Section 2, so in this section, we jump directly into the formal description.

We first state some natural assumptions on the underlying PIR scheme. Later in Appendix B, we show that these natural assumptions can actually be removed, i.e., we can generalize this balancing technique for any PIR scheme whose upload bandwidth is smaller than the download bandwidth.

Natural assumptions on the underlying PIR scheme. We assume that for a "natural" S-server (preprocessing) PIR scheme, the preprocessing algorithm and response algorithm are deterministic and identical for all servers, and the distribution of the messages sent to all servers are identical. More formally, we assume the following:

Assumption 7.1. 1. Each server $s \in [\![S]\!]$ uses same deterministic preprocessing algorithm $\widetilde{\mathsf{DB}} \leftarrow \mathbf{Preproc}(\mathsf{DB})$ and response algorithm $\mathbf{Answer}(\widetilde{\mathsf{DB}}, Q_s)$.

2. For any $s_1, s_2 \in [\![S]\!]$, the distributions $\{Q_{s_1} : (Q_0, \ldots, Q_{S-1}) \leftarrow \mathbf{Query}(n, 0)\}$ and $\{Q_{s_2} : (Q_0, \ldots, Q_{S-1}) \leftarrow \mathbf{Query}(n, 0)\}$ are identical.

The above guarantees that if the desired index is 0, then the query message is identically distributed for all servers. Together with the PIR's security property, it also implies that the distribution of the query message is identical for all servers no matter what index is queried.

Indeed, to the best of our knowledge, all known S-server PIR schemes, including the new schemes proposed in our paper, satisfy Assumption 7.1. We also observe that these assumptions can be removed — see Appendix B for detailed proof.

7.1 Construction

Parameters and notations. We will choose the following parameters.

- Let $\mathsf{PIR} = (\mathsf{PIR}.\mathbf{Preproc}, \mathsf{PIR}.\mathbf{Query}, \mathsf{PIR}.\mathbf{Answer}, \mathsf{PIR}.\mathbf{Recons})$ be a PIR scheme with global preprocessing that satisfies the aforementioned natural assumptions.
- Suppose that the n-bit database is partitioned into $B := n^{1-\mu}$ blocks each with n^μ bits, we use the notation DB_j to represent the j-th block of database. Without loss of generality, we assume that $B := n^{1-\mu}$ is an integer.

Balancing technique. We construct a new PIR scheme that makes blackbox calls to the underlying PIR.

- $\mathbf{Preproc}$: For each block j, each server performs $\widetilde{\mathsf{DB}}_j \leftarrow \mathsf{PIR}.\mathbf{Preproc}(\mathsf{DB}_j)$ to obtain an encoded database of the block and stores it.
- $\mathbf{Query}$: Let $i \in [\![n]\!]$ be the queried index, and $r = \lfloor i/n^\mu \rfloor$ be the block where i resides.

  For block $j = r$, client performs actual query algorithm of index i to obtain

  $$st_j, Q_{j,0}, \ldots, Q_{j,S-1} \leftarrow \mathsf{PIR}.\mathbf{Query}(n^\mu, i \bmod n^\mu)$$

  For other blocks $j \neq r$, client simply performs a dummy query:

  $$st_j, Q_{j,0}, \ldots, Q_{j,S-1} \leftarrow \mathsf{PIR}.\mathbf{Query}(n^\mu, 0)$$

  Then, the client randomly picks $b_0, b_1, \ldots, b_{B-1} \in \{0, 1\}$ and prepares the query messages:

  For $s = 0$, the client sets

  $$\vec{m}_{j,s} = (Q_{j,0}, b_j)$$

  And for other servers $s \in [\![S]\!]$, the client sets

  $$\vec{m}_{j,s} = \begin{cases} (Q_{j,0}, b_j) & \text{if } j \neq r \\ (Q_{j,s}, 1 - b_j) & \text{if } j = r \end{cases}$$

  The client then sends $(\vec{m}_{0,s}, \ldots, \vec{m}_{B-1,s})$ to each server $s \in [\![S]\!]$ and stores private state $st = (st_r, b_r)$.
- $\mathbf{Answer}$: The s-th server parses the message received from the client as $(Q'_{0,s}, b'_{0,s}, \ldots, Q'_{B-1,s}, b'_{B-1,s})$.

  For each block j, it computes $\mathsf{ans}_{j,s} = \mathsf{PIR}.\mathbf{Answer}(\widetilde{\mathsf{DB}}_j, Q'_{j,s})$.

  Then, for every block j, depending on the control bit $b'_{j,s}$, the server accumulates the response messages for block j into one of two slots, denoted $\mathsf{sum}_{s,0}$ and $\mathsf{sum}_{s,1}$, respectively:

  $$\mathsf{sum}_{s,0} = \bigoplus_{j=0}^{B-1} \mathsf{ans}_{j,s}(1 - b'_{j,s})$$

and

$$\mathsf{sum}_{s,1} = \bigoplus_{j=0}^{B-1} \mathsf{ans}_{j,s} b'_{j,s}$$

Finally, it sends back $\mathsf{sum}_{s,0}$ and $\mathsf{sum}_{s,1}$ to client.

- **Recons**: Parse st as (st_r, b_r). The client first extracts all S answers (of the underlying PIR) for the relevant block r:

$$\mathsf{ans}_{r,s} = \begin{cases} \mathsf{sum}_{s,1-b_r} \bigoplus \mathsf{sum}_{0,1-b_r} & \text{if } s \neq 0 \\ \mathsf{sum}_{s,b_r} \bigoplus \mathsf{sum}_{1,b_r} & \text{if } s = 0 \end{cases}$$

Then, it reconstructs $\mathsf{DB}[i]$ by applying the reconstruction algorithm of the underlying PIR:

$$\mathsf{DB}[i] = \mathsf{PIR}.\mathbf{Recons}(st_r, \mathsf{ans}_{r,0}, \ldots, \mathsf{ans}_{r,S-1})$$

7.2 Proof of Correctness

It suffices to show that client successfully extracts $\mathsf{ans}_{r,s} = \mathsf{PIR}.\mathbf{Answer}(\widetilde{\mathsf{DB}}_r, Q'_{r,s}) = \mathsf{PIR}.\mathbf{Answer}(\widetilde{\mathsf{DB}}_r, Q_{r,s})$ for each server $s \in [\![S]\!]$ since rest of proof immediately follows from correctness of the underlying PIR. Since the S-server PIR scheme PIR is natural (Assumption 7.1) and each server receives same query messages for all blocks $j \neq r$, hence they must compute same response messages $\mathsf{ans}_{j,\cdot}$ for such blocks.

Formally, we have

$$\begin{aligned} \mathsf{sum}_{s,b'_{r,s}} &= \mathsf{PIR}.\mathsf{ans}_{r,s} \bigoplus \mathsf{noise}_{b'_{r,s}} \\ \mathsf{sum}_{s,1-b'_{r,s}} &= \mathsf{noise}_{1-b'_{r,s}} \end{aligned}$$

where

$$\mathsf{noise}_b := \bigoplus_{j=0, j\neq r}^{B-1} \mathsf{PIR}.\mathbf{Answer}(\widetilde{\mathsf{DB}}_j, Q_{j,0}) \mathbf{1}_{b_j = b}$$

Observe that when $s \neq 0$, $b'_{r,s} = 1 - b_r$ and $\mathsf{sum}_{s,b_r} = \mathsf{noise}_{b_r}$; when $s = 0$, $b'_{r,s} = b_r$ and $\mathsf{sum}_{s,1-b_r} = \mathsf{noise}_{1-b_r}$. Thus for $s \neq 0$,

$$\begin{aligned} \mathsf{ans}_{r,s} &= \mathsf{sum}_{s,b'_{r,s}} \bigoplus \mathsf{noise}_{b'_{r,s}} \\ &= \mathsf{sum}_{s,1-b_r} \bigoplus \mathsf{noise}_{1-b_r} \\ &= \mathsf{sum}_{s,1-b_r} \bigoplus \mathsf{sum}_{0,1-b_r} \end{aligned}$$

for $s = 0$,

$$\begin{aligned} \mathsf{ans}_{r,s} &= \mathsf{sum}_{s,b'_{r,s}} \bigoplus \mathsf{noise}_{b'_{r,s}} \\ &= \mathsf{sum}_{s,b_r} \bigoplus \mathsf{noise}_{b_r} \\ &= \mathsf{sum}_{s,b_r} \bigoplus \mathsf{sum}_{1,b_r} \end{aligned}$$

7.3 Proof of Security

Observe that $b_j \xleftarrow{\$} \{0,1\}$ is randomly generated for each block j, thus $b'_{j,s}$ is also randomly distributed in $\{0,1\}$ for each server s. We claim that $Q'_{j,s}$ is also randomly distributed: for $j \neq r$ we have

$$\begin{aligned}\{Q'_{j,s}\} &\equiv \{Q_{j,0} : (Q_{j,0}, \ldots, Q_{j,S-1}) \leftarrow \mathsf{PIR}.\mathbf{Query}(n^\mu, 0)\} \\ &\equiv \{Q_{j,s} : (Q_{j,0}, \ldots, Q_{j,S-1}) \leftarrow \mathsf{PIR}.\mathbf{Query}(n^\mu, 0)\}\end{aligned}$$

where the second equation follows from Assumption 7.1; and for $j = r$

$$\begin{aligned}\{Q'_{j,s}\} &\equiv \{Q_{j,s} : (Q_{j,0}, \ldots, Q_{j,S-1}) \leftarrow \mathsf{PIR}.\mathbf{Query}(n^\mu, i \bmod n^\mu)\} \\ &\equiv \{Q_{j,s} : (Q_{j,0}, \ldots, Q_{j,S-1}) \leftarrow \mathsf{PIR}.\mathbf{Query}(n^\mu, 0)\}\end{aligned}$$

7.4 Efficiency

- **Bandwidth:** Assume the per-server upload and download bandwidth of PIR is bounded by $C_{\text{up}}(n)$ and $C_{\text{down}}(n)$, respectively.

  For each server s, it receives a query message $Q'_{j,s}$ and a bit $b'_{j,s}$ for each block j, thus in total takes upload bandwidth $O(n^{1-\mu} C_{\text{up}}(n^\mu))$, and it sends back two response messages which takes download bandwidth $O(C_{\text{down}}(n^\mu))$. Hence the total bandwidth is $O(n^{1-\mu} C_{\text{up}}(n^\mu) + C_{\text{down}}(n^\mu))$.
- **Server computation:** Assume the per-server computation of PIR is bounded by $T_{\text{answer}}(n)$, then each server needs to perform $\mathsf{PIR}.\mathbf{Answer}$ operation for each block j and compute the XOR sum of each response messages, in total takes server computation $O(n^{1-\mu} T_{\text{answer}}(n^\mu))$.
- **Client computation:** Assume $\mathsf{PIR}.\mathbf{Query}$ operation takes time $T_{\text{query}}(n)$ and $\mathsf{PIR}.\mathbf{Recons}$ operation takes time $T_{\text{recons}}(n)$. The client needs to compute $\mathsf{PIR}.\mathbf{Query}$ for every blocks but only perform $\mathsf{PIR}.\mathbf{Recons}$ once, in total takes client computation $O(n^{1-\mu} T_{\text{query}}(n^\mu) + T_{\text{recons}}(n^\mu))$.
- **Server space and preprocessing time:** Assume the server space and preprocessing time of PIR is bounded by $M(n)$ and $T_{\text{preproc}}(n)$, then clearly the new scheme takes server space $O(n^{1-\mu} M(n^\mu))$ and preprocessing time $O(n^{1-\mu} T_{\text{preproc}}(n^\mu))$.

In conclusion, we have

Lemma 7.2. *Suppose there exists an S-server PIR scheme satisfying Assumption 7.1 in which achieves $C_{\text{up}}(n)$ per-server upload bandwidth, $C_{\text{down}}(n)$ download bandwidth, $T_{\text{answer}}(n)$ per-server computation, $T_{\text{query}}(n)$ $\mathbf{Query}$ operation complexity per query and $T_{\text{recons}}(n)$ $\mathbf{Recons}$ operation complexity per query, with $M(n)$ server storage and $T_{\text{preproc}}(n)$ preprocessing time. Then for any $0 < \mu \leq 1$, there exists an S-server PIR scheme satisfying Assumption 7.1, it can achieve $O(n^{1-\mu} C_{\text{up}}(n^\mu) + C_{\text{down}}(n^\mu))$ per-server bandwidth, $O(n^{1-\mu} T_{\text{answer}}(n^\mu))$ per-server computation, $O(n^{1-\mu} T_{\text{query}}(n^\mu) + T_{\text{recons}}(n^\mu))$ client computation per query, with $O(n^{1-\mu} M(n^\mu))$ server storage and $O(n^{1-\mu} T_{\text{preproc}}(n^\mu))$ preprocessing time.*

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.

[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.

[BGG+24] Vishwas Bhargava, Sumanta Ghosh, Zeyu Guo, Mrinal Kumar, and Chris Umans. Fast multivariate multipoint evaluation over all finite fields. *Journal of the ACM*, 71(3):1–32, 2024.

[BGKM22] Vishwas Bhargava, Sumanta Ghosh, Mrinal Kumar, and Chandra Kanta Mohapatra. Fast, algebraic multivariate multipoint evaluation in small characteristic and applications. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 403–415, New York, NY, USA, 2022. Association for Computing Machinery.

[BIM04] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the Servers' Computation in PrivateInformation Retrieval: PIR with Preprocessing. *Journal of Cryptology*, 17(2):125–151, March 2004.

[BJPY18] Elette Boyle, Abhishek Jain, Manoj Prabhakaran, and Ching-Hua Yu. The bottleneck complexity of secure multiparty computation. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, pages 24–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

[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.

[Has36] Helmut Hasse. Theorie der höheren differentiale in einem algebraischen funktionenkörper mit vollkommenem konstantenkörper bei beliebiger charakteristik. *Journal für die reine und angewandte Mathematik*, 175:50–54, 1936.

[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.

[Kop13] Swastik Kopparty. Some remarks on multiplicity codes. *Discrete Geometry and Algebraic Combinatorics*, 625(155-176):1–1, 2013.

[KSY14] Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. *Journal of the ACM (JACM)*, 61(5):1–20, 2014.

[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.

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

[LLFP24] Arthur Lazzaretti, Zeyu Liu, Ben Fisch, and Charalampos Papamanthou. Multi-server doubly efficient PIR. Cryptology ePrint Archive, Paper 2024/829, 2024. https://eprint.iacr.org/2024/829.

[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/.

[SWZ24] Jaspal Singh, Yu Wei, and Vassilis Zikas. Information-theoretic multi-server private information retrieval with client preprocessing. In *TCC*, 2024.

[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.

[Y+12] Sergey Yekhanin et al. Locally decodable codes. *Foundations and Trends® in Theoretical Computer Science*, 6(3):139–255, 2012.

[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 Fast polynomial evaluation algorithm from [BGG+24]

We present the fast polynomial evaluation algorithm from [BGG+24], where the depth of the recursion is just 2. We re-write the algorithm from [BGG+24] as one with a preprocessing phase followed by an online phase. The polynomial $f(\vec{X}) \in \mathbb{F}_q[\vec{X}]$ is a m-variate polynomial where the individual degree of the variables is at most d'. It is to be evaluated at a point $\vec{a} \in \mathbb{F}_q^m$.

Preprocessing algorithm. In order to define $\mathsf{PolyPreprocess}(f)$, we define a helper algorithm $\mathsf{PolyPreprocessHelper}$ which is recursive. It takes as input parameters f, r, s, t: its goal is to preprocess the polynomial f for fast evaluation in $\mathbb{Z}_{r^s}$. t is a parameter that denotes the depth of recursion. The algorithm $\mathsf{PolyPreprocess}(f)$ runs $\mathsf{PolyPreprocessHelper}(f, q, 1, 2)$ and outputs everything that $\mathsf{PolyPreprocessHelper}$ stores as $\widetilde{f}$.

$\mathsf{PolyPreprocessHelper}(f, r, s, t)$ is defined as follows.

1. If $t = 0$:
   (a) For $\vec{e} \in \mathbb{N}^m$ with $|\vec{e}|_1 < s$:
      i. compute and store $f_{\vec{e}}(\vec{X}) = \overline{\partial}^{\vec{e}} \circ f(\vec{X}) \in \mathbb{Z}_{r^s}[\vec{X}]$
      ii. compute and store $f_{\vec{e}}(\vec{b})$ for $\vec{b} \in [\![r]\!]^m$ where $[\![r]\!]$ is identified with a subset of $\mathbb{Z}_{r^s}$ via $i \mapsto i + r^s\mathbb{Z}$
      iii. Return
2. For all $\vec{e} \in \mathbb{N}^m$, with $|\vec{e}|_1 < s$, compute and store $f_{\vec{e}}(\vec{X}) := \overline{\partial}^{\vec{e}} \circ f(\vec{X})$, and then compute a lift $\widetilde{f}_{\vec{e}} \in \mathbb{Z}[\vec{X}]$ of $f_{\vec{e}}$ with coefficient in $[\![r^s]\!]$.
3. Let $M := (d'+1)(r-1)^{d'+1}$. Find primes $p_1 < p_2 < \ldots < p_k \leq 16 \log M$ such that $\prod_{j=1}^{k} p_j > M$.
4. For $j \in [k]$:
   (a) For $\vec{e} \in \mathbb{N}^m$ with $|\vec{e}|_1 < s$:
      i. compute and store $f_{\vec{e},j}(\vec{X}) := \widetilde{f}_{\vec{e}}(\vec{X}) \bmod p_j^m \in \mathbb{Z}_{p_j^m}[\vec{X}]$.
      ii. Invoke $\mathsf{PolyPreprocessHelper}(f_{\vec{e},j}, p_j, m, t-1)$

Online algorithm. In order to define EvalPoly we define the following recursive helper algorithm OnlineHelper. It takes as input parameters $f, \vec{a}, r, s, t$ where $f \in \mathbb{Z}_{r^s}[X_1, \ldots, X_m]$, $\vec{a} \in [\![r^s]\!]^m$. It has access to the data structure $\widetilde{f}$ which is global. Its goal is to evaluate the polynomial f at $\vec{a}$ in $\mathbb{Z}_{r^s}$. t is a parameter that denotes the depth of recursion. The algorithm $\mathsf{EvalPoly}(\widetilde{f}, \vec{a})$ runs $\mathsf{OnlineHelper}(f, \vec{a}, q, 1, 2)$ and returns whatever OnlineHelper returns.

$\mathsf{OnlineHelper}(f, \vec{a}, r, s, t)$ is defined as follows.

1. If $t = 0$:
   (a) Compute $\vec{b} \in [\![r]\!]^m$ such that the coordinates of $\vec{b}$ are the remainders of the corresponding coordinates of $\vec{a}$ modulo r
   (b) Compute and return $f(\vec{a}) = \sum_{\vec{e} \in \mathbb{N}^m : |\vec{e}|_1 < s} \underbrace{f_{\vec{e}}(\vec{b})}_{\text{computed from preprocessing}} \cdot (\vec{a} - \vec{b})^{\vec{e}}$
2. Compute $\widetilde{\vec{a}} \in [\![r]\!]^m$ such that the coordinates of $\widetilde{\vec{a}}$ are the remainders of the corresponding coordinates of $\vec{a}$ modulo r. Compute $\vec{b} := \widetilde{\vec{a}} \bmod r^s$.
3. Let primes $p_1, p_2, \ldots, p_k$ be those computed in the preprocessing phase such that $\prod_{j=1}^{k} p_j > M$.
4. For $j \in [k]$:
   (a) Compute $\vec{a}_j := \widetilde{\vec{a}} \bmod p_j^m$
   (b) For $\vec{e} \in \mathbb{N}^m$, with $|\vec{e}|_1 < s$, compute $f_{\vec{e},j}(\vec{a}_j) = \mathsf{OnlineHelper}(f_{\vec{e},j}, \vec{a}_j, p_j, m, t-1)$
5. For $\vec{e} \in \mathbb{N}^m : |\vec{e}|_1 < s$, let $f_{\vec{e}}$ be as defined during preprocessing. Compute $\widetilde{f}_{\vec{e}}(\widetilde{\vec{a}})$ as the unique $Q \in [\![\prod_{j \in [k]} p_j^m]\!]$ such that $Q \bmod p_j^m = f_{\vec{e},j}(\vec{a_j})$ for $j \in [k]$.
6. Compute $f_{\vec{e}}(\vec{b}) = \widetilde{f}_{\vec{e}}(\widetilde{\vec{a}}) \bmod r^s$
7. Compute $f(\vec{a}) = \sum_{\vec{e} \in \mathbb{N}^m, |\vec{e}|_1 < s} f_{\vec{e}}(\vec{b}) \cdot (\vec{a} - \vec{b})^{\vec{e}}$ and return it.

Correctness.

We first argue that if all the coordinates of $\vec{a} - \vec{b}$ are mulitple of r, then in $\mathbb{Z}_{r^s}$, we have that

$$f(\vec{a}) = \sum_{\vec{e} \in \mathbb{N}^m : |\vec{e}| < s} \overline{\partial}^{\vec{e}} \circ f(\vec{b}) \cdot (\vec{a} - \vec{b})^{\vec{e}} \,.$$

From the definition of Hasse derivatives, we have that for any $\vec{a}, \vec{b} \in \mathbb{N}^m$

$$f(\vec{a}) = f(\vec{b} + (\vec{a} - \vec{b})) = \sum_{\vec{e} \in \mathbb{N}^m} \overline{\partial}^{\vec{e}} \circ f(\vec{b}) \cdot (\vec{a} - \vec{b})^{\vec{e}} \,.$$

If all the coordinates of $\vec{a} - \vec{b}$ are multiples of r, we have that $(\vec{a} - \vec{b})^{\vec{e}} = 0$ in $\mathbb{Z}_{r^s}$ for all $\vec{e} : |\vec{e}| \geq s$. Therefore, it follows that

$$f(\vec{a}) = \sum_{\vec{e} \in \mathbb{N}^m : |\vec{e}| < s} \overline{\partial}^{\vec{e}} \circ f(\vec{b}) \cdot (\vec{a} - \vec{b})^{\vec{e}} \,.$$

This fact implies that $\mathsf{OnlineHelper}(f, \vec{a}, r, s, 0)$ computes $f(\vec{a})$ in $\mathbb{Z}_{r^s}$ correctly.

We will now argue that $\mathsf{EvalPoly}(\widetilde{f}, \vec{a})$ returns $f(\vec{a})$ when $\widetilde{f}$ was returned by $\mathsf{PolyPreprocess}(f)$. When OnlineHelper is invoked with $(f, \vec{a}, q, 1, 2)$, since $s = 1$, the only vector $\vec{e} \in \mathbb{N}^m : |\vec{e}|_1 < 1$ is the zero

vector. So $\mathsf{OnlineHelper}(f_{\vec{0},j}, \vec{a}, p_j, m, 1)$ gets invoked for $j \in [k]$. Assuming these return $f_{\vec{0},j}(\vec{a}) \bmod p_j^m = f(\vec{a}) \bmod p_j^m$, coupled with the fact that $\widetilde{f}_{\vec{0}}(\vec{a}) \leq M^m$ (we argue this shortly), it follows from the correctness of Chinese remainder theorem that $f(\vec{a})$ is returned. We are that $\widetilde{f}_{\vec{0}}(\vec{a}) \leq M^m$. Since f is a m-variate polynomial with individual degree at most d', there are at most $(d'+1)^m$ monomials. Since $\vec{a} \in \mathbb{F}_q^m$, the maximum value of a monomial in $f(\vec{a})$ is at most $(q-1) \cdot (q-1)^{md'}$. Therefore, the maximum value of $\widetilde{f}_{\vec{0}}(\vec{a})$ is $(d'+1)^m \cdot (q-1)^{md'+1}$. Since, we have that $\prod_{j \in [k]} p_j^m > M = (d'+1)^m \cdot (q-1)^{md'+m} \geq (d'+1)^m \cdot (q-1)^{md'+1}$, it follows that $\widetilde{f}_{\vec{0}}(\vec{a}) \in [\![\prod_{j \in [k]} p_j^m]\!]$.

We can similarly show that $\mathsf{OnlineHelper}(f_{\vec{0},j}, \vec{a}, p_j, m, 1)$ returns $f(\vec{a}) \bmod p_j^m$ using the correctness of the Chinese remainder theorem and the fact that $\mathsf{OnlineHelper}(f, \vec{a}, r, s, 0)$ computes $f(\vec{a})$ in $\mathbb{Z}_{r^s}$ correctly (which we proved).

Efficiency. We analyze the running time and the storage of the preprocessing algorithm first. We write out the running times of the steps of PolyPreprocessHelper and then put it all together.

- For $t = 0$, the loop runs for $\binom{m+s-1}{s-1}$ times. The first step in the loop takes time $\mathsf{poly}(m, d', \log r)$, while the second step takes time r^m.
- For all $\vec{e} \in \mathbb{N}^m$, with $|\vec{e}_1| < s$, computing $\widetilde{f}_{\vec{e}}$ takes time $\binom{m+s-1}{s} \cdot \mathsf{poly}(m, d', \log r)$ time
- Computing M takes time $\mathsf{poly}(d', \log r)$ and the primes can be computed using sieve of Eratosthenes [], which takes time $\widetilde{O}(\log M) \leq O(\mathsf{poly}(d', \log r))$.
- The loop in step 4a has $\binom{m+s-1}{s-1}$ iterations and its first step takes time $\mathsf{poly}(m, d', \log r)$.

Since the preprocessing phase runs $\mathsf{PolyPreprocessHelper}(f, q, 1, 2)$, putting it together, the total runtime and storage required by the preprocessing algorithm is at most

$$O\left((16d'(\log d' + \log\log q))^m \cdot \mathsf{poly}(m, d', \log q)\right) .$$

We obtain the above by simply plugging in the running time of the steps in the execution of $\mathsf{PolyPreprocessHelper}(f, q, 1, 2)$ and additionally using the fact that $\binom{m+m-1}{m} \leq 4^m$.

We now analyze the running time of the online algorithm. We write out the running times of the steps of OnlineHelper and then put it all together.

- For $t = 0$, the running time is dominated by computing the summation in step (b)- it takes time $\binom{m+s-1}{s-1}\mathsf{poly}(m, d', \log r)$.
- Step 2 takes time $\mathsf{poly}(m, d', \log r)$
- Step 3 just takes constant time, since the algorithm just looks it up from the preprocessing
- The loop in step 4 runs $O(\log M)$ iteration. Step a) takes time $\mathsf{poly}(m, \log r)$, while step b has $\binom{m+s-1}{s-1}$ iterations each of which make a recurive call
- Step 5 takes times $\mathsf{poly}(m, d', \log r)$
- Step 6 takes time $\mathsf{poly}(m, d', \log r)$
- Step 7 takes time $\binom{m+s-1}{s-1}\mathsf{poly}(m, d', \log r)$

Since the online phase runs $\mathsf{OnlineHelper}(f, \vec{a}, q, 1, 2)$, putting it together, the total runtime of the online algorithm is at most

$$O\left(16^m \cdot \mathsf{poly}(m, d', \log q)\right) .$$

We obtain the above by simply plugging in the running time of the steps in the execution of $\mathsf{OnlineHelper}(f, \vec{a}, q, 1, 2)$ and additionally using the fact that $\binom{m+m-1}{m} \leq 4^m$.

So, we get the following theorem.

Theorem A.1 (Polynomial preprocessing theorem from [BGG+24]). *For a m-variate polynomial $f : \mathbb{F}_q^m \to \mathbb{F}_q$ with individual degree $\leq d'$, over some finite field $\mathbb{F}_q$, then there exists algorithms* PolyPreprocess, EvalPoly *such that*

- *The runtime of* $\mathsf{PolyPreprocess}(f)$ *is* $(16d'(\log d' + \log\log q)^m \cdot \mathsf{poly}(m, d', \log q)$
- *Let* $\widetilde{f} \leftarrow \mathsf{PolyPreprocess}(f)$*, for any* $\vec{x} \in \mathbb{F}_q^m$, $\mathsf{EvalPoly}(\widetilde{f}, \vec{x}) = f(\vec{x})$ *and the runtime of* $\mathsf{EvalPoly}(\widetilde{f}, \vec{x})$ *is* $16^m \cdot \mathsf{poly}(m, d', \log q)$.

B Removing the Natural Assumptions for Our Balancing Technique

The generic balancing technique earlier requires some natural assumptions on the underlying PIR scheme. In this section, we show that these natural assumptions can be removed. First, in Appendix B.1, we show how to transform any PIR scheme to one that satisfies the natural assumptions with an S factor blowup. Next, in Appendix B.2, we show that we can save the S-factor blowup by not going through the "arbitrary to natural" transformation of Appendix B.1. In fact, we can improve our balancing technique of Section 7 to directly work on top of an arbitrary PIR scheme.

Throughout the section, we assume that the underlying PIR scheme has a deterministic server-side algorithm. This assumption is for free as long as the PIR scheme is perfectly correct, since the server can always just fix the random coins.

B.1 Compiling Any PIR to a Natural One with S Factor Blowup

Here we present a construction in which given any S-server PIR scheme (where the server-side algorithm is deterministic), we can transform it into one that additionally satisfies Assumption 7.1, with S factor blowup.

B.1.1 Construction

Parameters and notations. For server number S, We will choose the following parameters.

- Let $\mathsf{PIR} = (\mathsf{PIR}.\mathbf{Preproc}_s, \mathsf{PIR}.\mathbf{Query}, \mathsf{PIR}.\mathbf{Answer}_s, \mathsf{PIR}.\mathbf{Recons})$ be a S-server PIR scheme with global preprocessing.

Compiler. We will use a simple circular parallel repetition strategy, that is, let each server simulates the behaviors of all servers in PIR with different queries for each.

- $\mathbf{Preproc}$: The same holds for each server $s \in [\![S]\!]$: for each $s' \in [\![S]\!]$, invoke $\widetilde{\mathsf{DB}}_{s'} = \mathsf{PIR}.\mathbf{Preproc}_{s'}(\mathsf{DB})$ and stores the preproccesing result $\{\widetilde{\mathsf{DB}}_{s'}\}_{s' \in [\![S]\!]}$.
- $\mathbf{Query}$: Given query index i, the client independently generates S queries $\mathsf{Que}_0, \mathsf{Que}_1, \ldots, \mathsf{Que}_{S-1}$about i. The j-th query Que_j is of the form

$$st_j, Q_{j,0}, \ldots, Q_{j,S-1} \leftarrow \mathsf{PIR}.\mathbf{Query}(n, i)$$

The client then sends $(Q_{(s'-s) \bmod S, s'})_{s' \in [\![S]\!]}$ to server s and stores private state st_0,

- **Answer**: The s-th server parses the message received from the client as $(Q'_{0,s}, \ldots, Q'_{S-1,s})$. The for each $s' \in [\![S]\!]$, it simulates the behavior of s'-th server of the underlying PIR scheme PIR taking message $Q'_{s',s}$ as input.

  Formally, for all $s' \in [\![S]\!]$ it computes and sends back

$$\mathsf{ans}_{s',s} = \mathsf{PIR}.\mathbf{Answer}_{s'}(\widetilde{\mathsf{DB}}_{s'}, Q'_{s',s})$$

- **Recons**: The client retrieves all servers' responses, then it only uses a diagonal part to reconstruct $\mathsf{DB}[i]$:

$$\mathsf{DB}[i] = \mathsf{PIR}.\mathbf{Recons}(st_0, \mathsf{ans}_{0,0}, \mathsf{ans}_{1,1}, \ldots, \mathsf{ans}_{S-1,S-1})$$

B.1.2 Proof of Correctness

Notice that $Q'_{s,s} = Q_{0,s}$ and indeed $\mathsf{ans}_{s,s} = \mathsf{PIR}.\mathbf{Answer}_s(\widetilde{\mathsf{DB}}_s, Q_{0,s})$, therefore the correctness simply follows from correctness of underlying PIR scheme PIR.

B.1.3 Proof of Security

We first prove the sematical security of our PIR scheme: each server $s \in [\![S]\!]$ receives a series of messages $Q'_{0,s}, \ldots, Q'_{S-1,s}$, and we claim these messages don't reveal nontrivial information of query index i:

$$\begin{aligned}
&\{Q'_{0,s}, \ldots, Q'_{S-1,s}\} \\
\equiv &\{Q_{u,s} : st_u, Q_{u,0}, \ldots, Q_{u,S-1} \leftarrow \mathsf{PIR}.\mathbf{Query}(n, i)\}_{s' \in [\![S]\!], u = (s'-s) \bmod S} \\
\equiv &\{Q_{s'} : st, Q_0, \ldots, Q_{S-1} \leftarrow \mathsf{PIR}.\mathbf{Query}(n, i)\}_{s' \in [\![S]\!]} \\
\equiv &\{Q_{s'} : st, Q_0, \ldots, Q_{S-1} \leftarrow \mathsf{PIR}.\mathbf{Query}(n, 0)\}_{s' \in [\![S]\!]}
\end{aligned}$$

where the last equation is due to the security of underlying PIR scheme PIR.

Observe that above argument also shows that the query message distributions of any two servers are identical, moreover clearly each pair of servers share same preprocessing and response algorithms, so the new scheme satisfies Assumption 7.1.

B.1.4 Efficiency

The new scheme can be viewed as parallel runs S independent instances of PIR, hence we have:

Lemma B.1. *Suppose there exists an S-server PIR scheme (with deterministic server-side algorithm), then there also exists an S-server PIR scheme satisfying Assumption 7.1 with S factor blowup of bandwidth, efficiency and server storage.*

Combining it with Lemma 7.2, we obtain:

Corollary B.2. *Suppose there exists an S-server PIR scheme with deterministic server-side algorithm in which achieves $C_{\mathrm{up}}(n)$ per-server upload bandwidth, $C_{\mathrm{down}}(n)$ download bandwidth, $T_{\mathrm{answer}}(n)$ per-server computation, $T_{\mathrm{query}}(n)$ $\mathbf{Query}$ operation complexity per query and $T_{\mathrm{recons}}(n)$ $\mathbf{Recons}$ operation complexity per query, with $M(n)$ server storage and $T_{\mathrm{preproc}}(n)$ preprocessing time. Then for any $0 < \mu \leq 1$, there exists an S-server PIR scheme satisfying Assumption 7.1, it can achieve $O((n^{1-\mu}C_{\mathrm{up}}(n^\mu) + C_{\mathrm{down}}(n^\mu))S)$ per-server bandwidth, $O(n^{1-\mu}T_{\mathrm{answer}}(n^\mu)S)$ per-server computation, $O((n^{1-\mu}T_{\mathrm{query}}(n^\mu) + T_{\mathrm{recons}}(n^\mu))S)$ client computation per query, with $O(n^{1-\mu}M(n^\mu)S)$ server storage and $O(n^{1-\mu}T_{\mathrm{preproc}}(n^\mu)S)$ preprocessing time.*

B.2 Balancing Technique for an Arbitrary PIR Scheme

In last section, we describe a method that transforms arbitrary PIR scheme to a "natural" version with S factor blowup, in the sense of making only blackbox calls it is optimal. However, we may find there still has some asymmetry in the construction: client only uses a diagonal part of response messages to reconstruct $\mathsf{DB}[i]$, since the choice of client is public and deterministic, it means that bulk of the responses are wasted.

Recall our goal is to apply balancing technique from arbitrary PIR scheme (with deterministic server-side algorithm), we may expect a careful construction will give better asymptotic complexity. In this section, we will describe a construction achieving constant overhead (that is, asymptotically best):

B.2.1 Construction

Parameters and notations. We will choose the following parameters.

- Let $\mathsf{PIR} = (\mathsf{PIR}.\mathbf{Preproc}_s, \mathsf{PIR}.\mathbf{Query}, \mathsf{PIR}.\mathbf{Answer}, \mathsf{PIR}.\mathbf{Recons})$ be a PIR scheme with deterministic server-side algorithm.
- Suppose that the n-bit database is partitioned into $B := n^{1-\mu}$ blocks each with n^μ bits, we use the notation DB_j to represent the j-th block of database. Without loss of generality, we assume that $B := n^{1-\mu}$ is an integer.
- We partition the S servers into $\lfloor S/2 \rfloor$ groups consisting of consecutive servers: all groups contains 2 consecutive servers except that the last group is constituted by the last 2 or 3 servers depending on parity of S.

Balancing technique. We construct a new PIR scheme that makes blackbox calls to the underlying PIR. Intuitively, this new scheme unifies ideas from Section 7 and Appendix B.1: for each small group with 2 or 3 members, client generates parallel repetition messages for each server inside this group, then it suffices to reconstruct the desired messages (on diagonal) by applying balancing techniques to this group. Since each group has only constant number of servers, the parallel repetition strategy will only incur a constant blowup.

- $\mathbf{Preproc}_s$: For each server s, it itemize each block j and each server s' where s' shares same group with it, and performs $\widetilde{\mathsf{DB}}_{j,s'} \leftarrow \mathsf{PIR}.\mathbf{Preproc}_{s'}(\mathsf{DB}_j)$ to obtain an encoded database of the block j and stores it.
- $\mathbf{Query}$: Let $i \in [\![n]\!]$ be the queried index, and $r = \lfloor i/n^\mu \rfloor$ be the block where i resides.

  For block $j = r$, client independently generates 3 queries of actual query index i. Formally, for each $k \in \{0, 1, 2\}$, client generates

  $$st_j^k, Q_{j,0}^k, Q_{j,1}^k, \ldots, Q_{j,S-1}^k \leftarrow \mathsf{PIR}.\mathbf{Query}(n^\mu, i \bmod n^\mu)$$

  For other blocks $j \neq r$ and each $k \in \{0, 1, 2\}$, client generates a dummy query of index 0:

  $$st_j^k, Q_{j,0}^k, Q_{j,1}^k, \ldots, Q_{j,S-1}^k \leftarrow \mathsf{PIR}.\mathbf{Query}(n^\mu, 0)$$

  Then, the client randomly picks $b_0, b_1, \ldots, b_{B-1} \in \{0, 1\}$ and prepares the query messages:

  For each server s resides in some group $\{2k, 2k+1\}$, if $s = 2k$ the client sets

  $$\vec{m}_{j,s} = (Q_{j,2k}^0, Q_{j,2k+1}^1, b_j)$$

And for the other server $s = 2k + 1$, the client sets

$$\vec{m}_{j,s} = \begin{cases} (Q^0_{j,2k}, Q^1_{j,2k+1}, b_j) & \text{if } j \neq r \\ (Q^1_{j,2k}, Q^0_{j,2k+1}, 1 - b_j) & \text{if } j = r \end{cases}$$

Suppose the last group contains 3 members $\{n-3, n-2, n-1\}$, the client will slightly change the query messages: for server $s = n - 3$, it sets

$$\vec{m}_{j,s} = (Q^0_{j,n-3}, Q^1_{j,n-2}, Q_{j,n-1}, b_j)$$

And for server $s = n - 2$ (the case of server $n - 1$ is symmetric, we omit it), client will set

$$\vec{m}_{j,s} = \begin{cases} (Q^0_{j,n-3}, Q^1_{j,n-2}, Q^2_{j,n-1}, b_j) & \text{if } j \neq r \\ (Q^2_{j,n-3}, Q^0_{j,n-2}, Q^1_{j,n-1}, 1 - b_j) & \text{if } j = r \end{cases}$$

The client then sends $(\vec{m}_{0,s}, \ldots, \vec{m}_{B-1,s})$ to each server $s \in [\![S]\!]$ and stores private state $st = (st^0_r, b_r)$.

- $\mathbf{Answer}_s$: For simplicity, we only discuss the case s belongs to some group $\{2k, 2k+1\}$ of size 2. The s-th server parses the message received from the client as

$$(Q'_{0,2k,s}, Q'_{0,2k+1,s}, b'_{0,s}, \ldots, Q'_{B-1,2k,s}, Q'_{B-1,2k+1,s}, b'_{B-1,s})$$

For each block j, it computes

$$\mathsf{ans}_{j,s} = (\mathsf{PIR}.\mathbf{Answer}_{2k}(\widetilde{\mathsf{DB}}_{j,2k}, Q'_{j,2k,s}), \mathsf{PIR}.\mathbf{Answer}_{2k+1}(\widetilde{\mathsf{DB}}_{j,2k+1}, Q'_{j,2k+1,s}))$$

Then, for every block j, depending on the control bit $b'_{j,s}$, the server accumulates the response messages for block j into one of two slots, denoted $\mathsf{sum}_{s,0}$ and $\mathsf{sum}_{s,1}$, respectively:

$$\mathsf{sum}_{s,0} = \bigoplus_{j=0}^{B-1} \mathsf{ans}_{j,s}(1 - b'_{j,s})$$

and

$$\mathsf{sum}_{s,1} = \bigoplus_{j=0}^{B-1} \mathsf{ans}_{j,s} b'_{j,s}$$

Finally, it sends back $\mathsf{sum}_{s,0}$ and $\mathsf{sum}_{s,1}$ to client.

- $\mathbf{Recons}$: Parse st as (st^0_r, b_r). The client first extracts all S answers (of the underlying PIR) of query Que^0_r similar to Section 7:

For group $\{2k, 2k+1\}$ of size 2, the client retrieves

$$\mathsf{ans}'_{r,2k} = (\mathsf{sum}_{2k,b_r} \bigoplus \mathsf{sum}_{2k+1,b_r})_0$$
$$\mathsf{ans}'_{r,2k+1} = (\mathsf{sum}_{2k+1,1-b_r} \bigoplus \mathsf{sum}_{2k,1-b_r})_1$$

Assuming the last group has size 3, the client retrieves

$$\mathsf{ans}'_{r,n-3} = (\mathsf{sum}_{n-3,b_r} \bigoplus \mathsf{sum}_{n-2,b_r})_0$$
$$\mathsf{ans}'_{r,n-2} = (\mathsf{sum}_{n-2,1-b_r} \bigoplus \mathsf{sum}_{n-3,1-b_r})_1$$
$$\mathsf{ans}'_{r,n-1} = (\mathsf{sum}_{n-1,1-b_r} \bigoplus \mathsf{sum}_{n-3,1-b_r})_1$$

Then, it reconstructs $\mathsf{DB}[i]$ by applying the reconstruction algorithm of the underlying PIR:

$$\mathsf{DB}[i] = \mathsf{PIR}.\mathbf{Recons}(st_r^0, \mathsf{ans}'_{r,0}, \ldots, \mathsf{ans}'_{r,S-1})$$

B.2.2 Proof of Correctness

The correctness proof is essentially same as Section 7.2, we omit here.

B.2.3 Proof of Security

Clearly $b'_{j,s}$ are always randomly distributed in $\{0,1\}$ for any choice of block r and server s. Notice that the query messages of each block j are independently generated, and for any block j each server s never receives query messages of same query twice, so the privacy of Q' can be deduced from analogous analysis as Section 7.3.

B.2.4 Efficiency

Comparing to scheme in Section 7, now each server needs to simulate all members of its group. Fortunately, each group has only constant size, thus the total blowup is also constant.

In conclusion, we have:

Lemma B.3. *Suppose there exists an S-server PIR scheme with deterministic server-side algorithm, in which achieves $C_{\mathrm{up}}(n)$ per-server upload bandwidth, $C_{\mathrm{down}}(n)$ download bandwidth, $T_{\mathrm{answer}}(n)$ per-server computation, $T_{\mathrm{query}}(n)$ **Query** operation complexity per query and $T_{\mathrm{recons}}(n)$ **Recons** operation complexity per query, with $M(n)$ server storage and $T_{\mathrm{preproc}}(n)$ preprocessing time. Then for any $0 < \mu \leq 1$, there exists an S-server PIR scheme achieving $O(n^{1-\mu}C_{\mathrm{up}}(n^\mu) + C_{\mathrm{down}}(n^\mu))$ per-server bandwidth, $O(n^{1-\mu}T_{\mathrm{answer}}(n^\mu))$ per-server computation, $O(n^{1-\mu}T_{\mathrm{query}}(n^\mu) + T_{\mathrm{recons}}(n^\mu))$ client computation per query, with $O(n^{1-\mu}M(n^\mu))$ server storage and $O(n^{1-\mu}T_{\mathrm{preproc}}(n^\mu))$ preprocessing time.*

This lemma slightly improves Corollary B.2 by an S factor in complexity. Also remind that assuming the determinacy of server-side algorithm is withou loss of generality, therefore we fully remove Assumption 7.1 from Lemma 7.2 while remains the asymptotic result.

C The Case of Polylogarithmically Many Servers

Beimel et al. [BIM04] achieve doubly-efficient PIR for the special case of polylogarimically many servers. In this section, we show that we can use our unified framework to match the result of Beimel et al. [BIM04]'s Theorem 4.9 through a different way of parametrization.

C.1 Construction

Parameters and notation. For database size n, we choose the following parameters:

- Let $\epsilon > 0$ be a constant, set $m = \lceil \epsilon \log n/(\log\log n)\rceil$, $d = \lceil n^{1/m}\rceil \leq \lceil \log^{1/\epsilon} n\rceil$ such that $d^m \geq n$.
- Let number of servers $S = S(n)$ be $md + 1 = O(\log^{1+1/\epsilon} n/\log\log n)$.
- We set q to be the smallest prime such that $q > S$, and will work on finite field $\mathbb{F}_q$. By Bertrand's postulate, $q \leq 2S$.

S-server PIR. Our S-server PIR works as follows. Different from all previous schemes, this scheme doesn't use derivatives. It can also be viewed as a special case of our generic scheme with $t = 1$, i.e., the server sends derivatives up to order zero.

- $\mathbf{Preproc}_s$: Encode database DB to m-variate polynomial F with individual degree $d = q - 1$. Concretely, we construct $E : [\![n]\!] \to \mathbb{F}_q^m$ be an *injective* index function, and recover F by interpolating on the set $\{\mathsf{DB}[i]\}_{i\in[\![n]\!]}$ using the techniques described by Lin et al. [LMW23]. Then each server s precomputes and stores $F(\vec{x})$ for all $\vec{x} \in \mathbb{F}_q^m$ with algorithm described in Lemma 4.7.

  Moreover, each server s individually picks a unique and **nonzero** element in $\mathbb{F}_q$ called λ_s and publishes it ($\lambda_1, \ldots, \lambda_{S-1}$ are public for all servers and client), and computes $w_s = l_s(0)$, where

$$l_s(\lambda) = \prod_{j=0, j\neq s}^{S-1} (\lambda - \lambda_s)(\lambda_j - \lambda_s)^{-1}$$

  is the s-th Lagrange basis polynomial.

- $\mathbf{Query}$: Given query index i, the client uniformly generates $\vec{v} \in \mathbb{F}_q^m$, and sets $\vec{u} = E(i)$.

  For $s \in [\![S]\!]$, the client sets

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

  The client sends $Q_s = \vec{z}_s$ to each server $s \in [\![S]\!]$.

- $\mathbf{Answer}_s$: The s-th server parses the message received from the client as a vector $\vec{z}_s$. It then sends back

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

  to the client.

- $\mathbf{Recons}$: Define univariate polynomial $f(\lambda) = F(\vec{u} + \lambda\vec{v})$, clearly $\vec{z}_s = \vec{u} + \lambda_s\vec{v} = f(\lambda_s)$ and $f(0) = F(\vec{u}) = \mathsf{DB}[i]$. Given the responses of all servers, the client computes:

$$\begin{aligned} f(0) &= \sum_{s=0}^{S-1} f(\lambda_s) l_s(0) \\ &= \sum_{s=0}^{S-1} F(\vec{z}_s) w_s \\ &= \sum_{s=0}^{S-1} \mathsf{ans}_s \end{aligned}$$

C.2 Proof of Correctness

The correctness of PIR scheme just follows from the correctness of Lagrange interpolation.

C.3 Proof of Security

The security proof is same as Section 4.2.2.

C.4 Efficiency

We now analyze the efficiency of our construction.

- **Bandwidth**: Each server receives a vector $\vec{z}_s \in \mathbb{F}_q^m$ and sends back $\mathsf{ans}_s \in \mathbb{F}_q$, thus the total bandwidth is $O(m \log q) = O(\epsilon \log n)$.
- **Server computation**: Since both $F(\vec{z}_s)$ and w_s are precomputed, the server computation is bounded by total bandwidth, that is, $O(\epsilon \log n)$.
- **Client computation**: Bandwidth is part of computation, which is $O(Sm \log q)$, and client needs to add up S elements in $\mathbb{F}_q$ (each takes time $O(\log q)$). Therefore, the total client computation is $O(Sm \log q + S \log q) = O(\epsilon^2 \log^{2+1/\epsilon} n / \log\log n)$.
- **Server space**: Each server should store $F(\vec{x})$ for all $\vec{x} \in \mathbb{F}_q^m$ and w_s (which takes only $\log q$ bits). There are q^m elements in $\mathbb{F}_q$, thus total storage is

$$\begin{aligned} & O(q^m \log q) \\ =& O(dm)^m \cdot \log q \\ =& d^m \cdot O(m)^m \cdot \log q \\ =& n^{1+\epsilon+O(\epsilon/\log\log n)} \end{aligned}$$

  where the first equation follows from the fact $q \le 2S = O(md)$.

- **Preprocessing time**: Since computing Lagrange polynomial only takes time $O(\mathsf{poly}(S, \log q)) = O(\mathsf{poly}\log n)$, the bottleneck is precomputing $F(\vec{x})$ for all $\vec{x} \in \mathbb{F}_q^m$. By Lemma 4.7, it takes time

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

  Since $\epsilon/\log\log n$ goes to 0 as n goes to infinity, we have

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

D Proof of Lemma 6.1

We now prove Lemma 6.1. By Stirling's approximation, we have

$$\binom{m}{\theta m} \ge \frac{2^{H(\theta)m}}{\sqrt{2\pi m\theta(1-\theta)}}(1-o(1)) \ge 2^{H(\theta)m - 0.5\log(m\theta(1-\theta)) - O(1)}(1-o(1))$$

Since $H(\theta) \ge \theta(1-\theta)$ for $\theta \in [0,1]$, the above is lower bounded by $2^{H(\theta)m - 0.5\log(H(\theta m)) - O(1)} \ge 2^{H(\theta)m(1-o(1))}$. To satisfy $\binom{m}{\theta m} > 0$, it suffices to set $m = \frac{\log n}{H(\theta)}(1+o(1))$ where $o(1)$ hides $O(\log\log n / \log n)$ terms.

D.1 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]. 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.