Information-Theoretic Multi-Server PIR with Global Preprocessing*

Ashrujit Ghoshal
CMU
aghoshal@andrew.cmu.edu

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

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

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

Elaine Shi†
CMU
runting@gmail.com

Abstract

We propose a new unified framework to construct multi-server, information-theoretic Private Information Retrieval (PIR) schemes that leverage global preprocesing to achieve sublinear computation per query. Despite a couple earlier attempts, our understanding of PIR schemes in the global preprocessing model remains limited, and so far, we only know a few sparse points in the broad design space. Our framework not only unifies earlier results in this space, but leads to several new results. First, we can improve the server space of the state-of-the-art scheme by a polynomial factor. Second, we can broaden the parameter space of known results, allowing a smooth tradeoff between bandwidth and computation. Third, while earlier schemes achieve better per-server bandwidth and computation as we add more servers, the server space actually grows w.r.t. the number of servers. We offer a new scalable family of schemes where the per-server bandwidth, computation, and space all decrease as we add more servers. This scalable family of schemes also implies the so-called "doubly efficient" PIR scheme with any super-constant number of servers, achieving $n^{1+o(1)}$ server space and preprocessing cost, and $n^{o(1)}$ bandwidth and computation per query.

1 Introduction

Private Information Retrieval (PIR), originally proposed by Chor et al. [CGKS95], allows a client to retrieve an entry from a public database stored on one or more server(s), without leaking its query to any individual server. PIR promises numerous applications such as private DNS [Fea, obl, SCV+21], privately checking whether one's password is in some leaked password database [hav, DRRT18], private contact discovery [sig], private web search [HDCG+23], and so on. In classical PIR schemes [CGKS95, Cha04, GR05, CMS99, CG97, KO97, Lip09, OS07, Gas04, BFG03, SC07, OG11, MCG+08, MG07, HHCG+23, MW22], the servers store the original database and there is no preprocessing. Unfortunately, Beimel, Ishai, and Malkin [BIM04] proved any classical PIR scheme (without preprocessing) suffers from a fundamental limitation, that is, every query must incur a linear (in the database size) amount of server computation. Intuitively, if there is some entry that the server does not look at to answer a client's query, then the client cannot be interested in that entry.

To scale PIR to large datasets, Beimel et al. [BIM04] introduced a new preprocessing model for PIR, and showed that with preprocessing, we can overcome the linear server computation barrier. In the *global* preprocessing model proposed by Beimel et al. [BIM04], the server computes and stores an encoded version of

*We would like to gratefully acknowledge Yuval Ishai and Henry Corrigan-Gibbs for suggesting the generic balancing technique described in Section 4. The version described in our paper is a slight improvement of their original idea.

†Author ordering is randomized.

the database which can be polynomial in size, and the same preprocessing is shared across all clients. Subsequent works have also considered a *client-specific* preprocessing model [CK20, CHK22, ZLTS23, LP23, LP22, SACM21, ZPSZ24, GZS24] where each client performs a separate preprocessing with the server (also called the subscription phase), and at the end of the preprocessing each client stores a hint that is related to the database. In comparison with client-specific preprocessing, the global preprocessing model enjoys some advantages. First, the same preprocessing can be amortized to an unbounded number of clients. In other words, the total preprocessing work and total space consumption do not depend on the number of clients. Second, for a dynamically evolving database, PIR with global preprocessing can easily be made dynamic using the standard hierarchical data structure by Bentley and Saxe [BS80], whereas with client-specific preprocessing, *every* client would have to update its hint for each update to the database [KCG21, HPPY24], and the costs can be significant for fast-evolving databases.

In this work, we focus on information-theoretic PIR schemes in the global preprocessing model, which is exactly the model considered by Beimel et al. [BIM04]. We focus on the setting of two or more servers which is necessary for achieving information-theoretic security due to well-known lower bounds [DMO00]. Since we focus on the global preprocessing model, we are particularly interested in PIR schemes that achieve *sublinear* computation per query.

Status quo and open questions. Despite earlier attempts [BIM04, WY05], our existing understanding of information-theoretic PIR in the global preprocessing model is rather limited. As we discuss below, so far, we only know a few sparse points in the entire design space; and numerous open questions remain.

First, in the *2-server* setting, Beimel et al. [BIM04] and the subsequent work of Woodruff and Yekhanin [WY05] showed that with a polynomial amount of server space, we can get PIR schemes with $O(n^{1/3})$ bandwidth and $n/\mathsf{poly}\log n$ computation per query, where n denotes the size of the database. The obvious disadvantage is that the $n/\mathsf{poly}\log n$ computation is only *slightly sublinear*, and it would be more desirable to get schemes whose computation is *significantly bounded away from linear*, e.g., in the n^δ regime for some constant $\delta \in (0,1)$. For this regime, the only known result is a scheme also proposed by Beimel et al. [BIM04], who showed how to achieve $n^{(1+\epsilon)/2}$ cost per query both in terms of bandwidth and server computation. Therefore, a natural question is the following:

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

Note that if we did not mind having linear server computation, it is known how to achieve bandwidth as small as $n^{o(1)}$ [DG16]. On the other hand, no scheme with less than $n^{1/2}$ bandwidth is known if we require the computation to be asymptotically less than $n/\mathsf{poly}\log n$.

More generally, in the S-server setting, no known scheme can achieve $o(n^{1/S})$ bandwidth if we require the per-query computation to be asymptotically less than $n/\mathsf{poly}\log n$. Beimel et al. [BIM04] showed how to attain $O(n^{(1+\epsilon)/S})$ bandwidth and computation; alternatively, they showed another scheme with $O(n^{1/(2S-1)})$ bandwidth but at the price of $O(n/\log^{2S-2} n)$ computation which is nearly linear.

> Open question 2: *Can we get an S-server information-theoretic PIR scheme with per-query bandwidth asymptotically less than $O(n^{1/S})$ and computation in the n^δ regime for some constant $\delta \in (0,1)$?*

So far, known results in the n^δ-computation regime [BIM04] allow the server to store a $\mathsf{poly}(n)$-sized encoding of the database. Prior works suffer from a large poly which may be prohibitive for large datasets. In fact, prior works did not even care about quantifying how large the poly is. Therefore, another natural question is the following:

Table 1: **2-server information-theoretic PIR**. The expression marked $\star$ assumes sufficiently large n and sufficiently small ϵ.

Scheme	Server Compute	Bandwidth	Server Space
[DG16]	$\geq n$	$n^{o(1)}$	0
[BIM04]	$n^{(1+\epsilon)/2}$	$n^{(1+\epsilon)/2}$	$n^{0.368\epsilon \cdot 2^{(1+o(1))/\epsilon}}$ $(\star)$
[BIM04]	$O(n/\log^2 n)$	$O(n^{1/3})$	$O(n^2)$
[WY05]	$n/\mathsf{poly}\log n$	$O(n^{1/3})$	$\mathsf{poly}(n)$
Corollary 1.4 $(\forall 1/3 \leq \alpha \leq 1/2)$	$n^{1-(1-\epsilon)\alpha}$	$n^{(1+\epsilon)\alpha}$	$n^{1-2\alpha+0.368\alpha\epsilon \cdot 2^{1+(1+o(1))/\epsilon}}$

Table 2: S**-server information-theoretic PIR.** The expression marked $\star$ and $\diamond$ assumes sufficiently large n and sufficiently small ϵ, the expression marked $\diamond$ additionally assumes that S is a prime, prime power, or sufficiently large.

Scheme	Server Compute	Bandwidth	Server Space
[BIM04]	$n^{(1+\epsilon)/S}$	$n^{(1+\epsilon)/S}$	$n^{0.368(S/\log S)\epsilon S^{(1+o(1))/\epsilon}}$ $(\star)$
[BIM04]	$O(n/\log^{2S-2} n)$	$O(n^{1/(2S-1)})$	$\mathsf{poly}(n)$ for $S = O(1)$
Theorem 1.1	$n^{(1+\epsilon)/S}$	$n^{(1+\epsilon)/S}$	$n^{0.368\epsilon S^{(1+o(1))/\epsilon}}$ $(\diamond)$
Theorem 1.3 $(\forall \frac{1}{S+1} \leq \alpha \leq \frac{1}{S})$	$n^{1-(S-1-\epsilon)\alpha}$	$n^{(1+\epsilon)\alpha}$	$n^{1-S\alpha+0.368\alpha\epsilon S^{1+(1+o(1))/\epsilon}}$ $(\diamond)$
Theorem 1.5	$O(n^{2/(\log S-1)} \log S)$	$O(n^{2/(\log S-1)} \log S)$	$\widetilde{O}(n^{1+2/(\log S-1)})$

Open question 3: *Can we asymptotically improve the server space for S-server PIR schemes with sublinear computation?*

Last but not the least, the earlier works of Beimel et al. [BIM04] and Woodruff and Yekhanin [WY05] focus on optimizing the bandwidth consumption. The show that we can have a family of schemes where the bandwidth and computation decrease w.r.t. the number of servers, unfortunately the server space increases with more servers. In this paper, we want to have a family of schemes that are *scalable* w.r.t. the number of servers, that is, we want the per-server bandwidth, computation, space and preprocessing cost to all decrease with the addition of more servers.

Open question 4: *Can we have a family of schemes that achieve scalability w.r.t. the number of servers S?*

1.1 Our Results and Contributions

In this work, we propose a new, unified framework for constructing multi-server information-theoretic PIR in the global preprocessing model. Using this unified framework, we can unify and asymptotically improve all prior and concurrent results on multi-server information-theoretic PIR with sublinear computation. Not only so, we can also generalize earlier and concurrent results to broader parameter regimes. Further, we can get a scalable family of multi-server PIR from this unified framework. Thus we provide an affirmative answer to all of the aforementioned questions. We now summarize our results and contributions.

Polynomial improvement in server space. First, we propose an improved framework for constructing S-server PIR such that we can match the bandwidth and computational overhead of Beimel et al. [BIM04], while *improving their server space by a polynomial factor*. Below, we first give the general version parametrized by an intermediate parameter θ, and then we interpret the parameters and compare with Beimel et al. [BIM04]. Henceforth, let $H(\cdot)$ be the binary entropy function.

Theorem 1.1 (Multi-server PIR with improved server space: general form)**.** *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 $\mathsf{poly}(n)$.*

In comparison, Beimel et al. [BIM04] achieves the same bandwidth and computation overhead, but their preprocessing cost and server space are $n^{(1+o(1))(S-1+H(\theta/S))/H(\theta)}$. Concretely, when $S = 2$, our scheme chooses $q = 2$ which means $\log q = S - 1$. Thus, our scheme has the same cost as Beimel et al. [BIM04] for $S = 2$. For any $S > 2$, our scheme achieves a polynomial factor saving in server space and preprocessing cost than Beimel et al. [BIM04]. In particular, we save a factor of $(S-1)/(\log S + 1)$ to $(S-1)/\log S$ in the exponent depending on S.

For sufficiently large S, n, and sufficiently small ϵ, it is possible to simplify the expressions in Theorem 1.1 through Taylor expansion, leading to the following corrolary:

Corollary 1.2 (Multi-server PIR with improved server space)**.** *For sufficiently large n and sufficiently small $\epsilon > 0$, there exists an S-server PIR scheme such that, which achieves $O(n^{(1+\epsilon)/S} \log S)$ per-server bandwidth, $O(n^{(1+\epsilon)/S} \log S)$ per-server computation and $O(n^{(1+\epsilon)/S} S \log^2 S)$ client computation per query, and the preprocessing time and server space are bounded by $n^{0.368\epsilon S^{(1+o(1))/\epsilon}}$ if S is either a prime, a prime power, or sufficiently large. For an arbitrary S, the server space and preprocessing time are bounded by $n^{0.4\epsilon S^{(1+o(1))/\epsilon}}$ for sufficiently large n and sufficiently small ϵ.*

In comparison, with sufficiently large S, n, and sufficiently small $\epsilon > 0$, Beimel et al. [BIM04] has $n^{0.368(S/\log S)\cdot\epsilon\cdot S^{(1+o(1))/\epsilon}}$ server space and preprocessing cost [1], and thus we save a factor of roughly $S/\log S$ in the exponent.

A general balancing technique. Chor et al. [CGKS95] showed that given a PIR scheme whose upload bandwidth is larger than the download bandwidth, we can apply a generic balancing trick to asymptotically reduce the total bandwidth. A restated version of this balancing technique can also be found in Lemma 4.4 of Beimel et al. [BIM04]. Unfortunately, this balancing trick no longer works when the underlying PIR has larger download bandwidth than upload bandwidth, which is often the case for PIR schemes with sublinear computation. In particular, the way Beimel et al. [BIM04] achieves sublinear computation is to ensure that the upload bandwidth (i.e., query message length) is $O(\log n)$. This way, each server can precompute all answers and simply look it up during the online query.

We propose a new generic balancing technique that works for the opposite scenario, i.e., when the underlying PIR has larger download bandwidth than upload bandwidth. Applying this new generic technique to the scheme of Beimel et al. [BIM04] or our earlier Theorem 1.1 both of which $n^{(1+\epsilon)/S}$ bandwidth and computation, we can make the bandwidth as small as $n^{(1+\epsilon)/(S+1)}$ with a slightly increased computational overhead of $n^{(2+\epsilon)/(S+1)}$. In fact, this balancing technique allows a smooth tradeoff curve between the bandwidth and computation, as stated in the following theorem:

[1] For both our scheme and Beimel et al. [BIM04], the $o(1)$ in the exponent of $S^{((1+o(1))/\epsilon}$ hides $o(\epsilon) + O(\log\log n/\log n)$ terms which makes them directly comparable. Also for this reason, we cannot absorb the $S/\log S$ part into the $o(1)$.

Theorem 1.3 (Minimizing the bandwidth through a balancing technique)**.** *Suppose that the number of servers S is a constant. For any $1/(S+1) \leq \alpha \leq 1/S$ and an arbitrarily small $\epsilon > 0$, there exists an information-theoretic S-server preprocessing PIR scheme with $n^{(1+\epsilon)\alpha}$ bandwidth and client computation, and $n^{1-(S-1-\epsilon)\alpha}$ server computation per query, assuming $\mathsf{poly}(n)$ amount of server space. Further, for sufficiently large n and sufficiently small ϵ, the server space and preprocessing cost is upper bounded by $n^{1-S\alpha+0.368\alpha\epsilon S^{1+(1+o(1))/\epsilon}}$ if S is either a prime, prime power, or sufficiently large; or bounded by $n^{1-S\alpha+\alpha 0.4\epsilon S^{1+(1+o(1))/\epsilon}}$ for any arbitrary S.*

For the special case when $S = 2$, the above Theorem 1.3 immediately gives rise to the following corollary.

Corollary 1.4 (2-server special case of Theorem 1.3)**.** *For any $1/3 \leq \alpha \leq 1/2$, for an arbitrarily small $\epsilon \in (0,1)$, there exists an information-theoretic 2-server preprocessing PIR scheme with $n^{(1+\epsilon)\alpha}$ bandwidth and client computation, and $n^{1-(1-\epsilon)\alpha}$ per-server computation, assuming $n^{1-2\alpha+0.368\alpha\epsilon\cdot 2^{1+(1+o(1))/\epsilon}}$ amount of server space.*

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

Achieving scalability with more servers. Just like prior work [BIM04], the schemes mentioned so far (Theorem 1.1, Corollary 1.2, Theorem 1.3, and Corollary 1.4) focus on optimizing the bandwidth while keeping the computation in the n^{δ} regime and the server space and preprocessing cost in the $\mathsf{poly}(n)$ regime. While their the per-server bandwidth and computation decrease with more servers, the server space and preprocessing costs actually grow with more servers.

We say that a scheme is *scalable* w.r.t. the number of servers if the per-server bandwidth, computation, space and preprocessing cost all decrease with the addition of more servers. Using the same framework we propose but relying on a different way of parameterization, we show how to get a new family of schemes that achieve scalability. Specifically, for any $S = \omega(1)$ number of servers, we can construct an S-server PIR scheme with $n^{o(1)}$ communication and computation per query, with $n^{1+o(1)}$ preprocessing cost and server space — a scheme with such performance is often referred to as a "doubly efficient" PIR [CHR17, BIPW17, LMW23]. In comparison with the concurrent work of Lazzaretti et al. [LLFP24], they require super-logarithmically many servers in order to achieve $n^{o(1)}$ communication and computation and $n^{1+o(1)}$ server storage simultaneously, whereas our framework can achieve this with only super-constant number of servers.

The following theorem captures our scalable family of multi-server PIR more generally:

Theorem 1.5 (Scalable family of multi-server PIR)**.** *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 $O(S\log S\cdot n^{2/(\log S-1)})$ client computation per query, with $\widetilde{O}(n^{1+2/(\log S-1)})$ preprocessing time and server storage where $\widetilde{O}(\cdot)$ hides a $\mathsf{poly}\log n$ multiplicative factor.*

It is not hard to see that for $S = 2^{o(\log n/\log\log n)}$, the term $n^{2/(\log S-1)}$ dominates the term $\log S$. In other words, as long as S is sufficiently smaller than n the per-server bandwidth and computation as well as server space and preprocessing costs all decrease as we add more servers. Since the client needs to receive messages from all S servers, we allow the client computation to have a factor that grows linearly with S.

Remark 1.6 (Comparison with the concurrent work of Lazzaretti et al. [LLFP24])**.** An initial version of our paper was eprinted on May 19, 2024, more than one week before Lazzaretti et al. [LLFP24]. In this new version, we further improved the server space by using a more generic balancing trick rather than a non-black balancing trick. We also added the scalable family of schemes using the same unified framework, which implies Lazzaretti et al. [LLFP24]'s "doubly efficient" notion with only super-constant number of servers, and without the use of a polynomial evaluation data structure. We sent this version of the paper (with minor editorial differences) to Lazzaretti et al. on June 13, 2024.

Finally, as a by-product, with the same unified 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 B).

1.2 Additional Related Work

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

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

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

2 Technical Roadmap

2.1 A Unified Framework for Constructing Multi-Server PIR

Many of the results stated earlier in Section 1.1 can be viewed as different instantiations of a new, unified framework. Beimel et al. [BIM04] suggest that we can construct multi-server PIR with sublinear computation with the following recipe. First, construct a classical PIR scheme with $O(\log n)$ per-server upload bandwidth (i.e., query message length). With such a scheme, each server can simply precompute and store all answers which takes $\mathsf{poly}(n)$ space, and during the online query, the server simply looks up the answer.

We start with the scheme of Woodruff and Yekhanin [WY05]. Unfortunately, Woodruff and Yekhanin's scheme does not enjoy succinct upload bandwidth. For this reason, they resort to a more complicated technique to achieve sublinear computation, and they can only achieve slightly sublinear (i.e., $n/\mathsf{poly}\log n$) computation. We devise new techniques to compress Woodruff and Yekhanin's upload bandwidth to logarithmic.

Background: the framework of Woodruff and Yekhanin. To understand our techniques, it helps to quickly review their scheme first, and see why they suffer from large upload bandwidth.

- *Encoding the database as a polynomial.* Woodruff and Yekhanin's scheme enocodes the database as an m-variate polynomial $F(x_1, \ldots, x_m)$ of individual degree 1 and total degree d, over some finite field $\mathbb{F}_q$. Each database index $i \in \{0, 1, \ldots, n-1\}$ can be uniquely encoded as a length-m vector denoted $E(i) \in \mathbb{F}_q^m$, and it is guaranteed that $\mathsf{DB}[i] = F(E(i))$. For this encoding to work, it must be that $\binom{m}{d} \geq n$.
- *Protocol.* To obtain $\mathsf{DB}[i]$, the client chooses a random vector $\vec{v}$ and sends S distinct points on the line $E(i) + \lambda\vec{v}$ to each of the S servers, by choosing $\lambda = \omega^0, \omega^1, \ldots, \omega^{S-1}$ respectively where ω is the S-th root of unity. When a server $s \in \{0, 1, \ldots, S-1\}$ receives a vector $\vec{z}_s = E(i) + \omega^s\vec{v}$, it responds with the polynomial F evaluated at $\vec{z}_s$, as well as all derivatives of F up to d/S-th order, evaluated also at $\vec{z}_s$.
- *Reconstruction.* The client then considers the polynomial F projected onto the line $E(i) + \lambda\vec{v}$, that is, $g(\lambda) := F(E(i) + \lambda\vec{v})$. $g(\lambda)$ is a polynomial in λ of the form $g(\lambda) = \sum_{k \leq d/S} c_k \lambda^{kS}$. In other words, only terms of form λ^j where j is a power of S survive in $g(\lambda)$. Using the answers from the servers and the chain rule for derivatives, the client can compute the derivatives $g(1), g'(1), g^{(2)}(1), \ldots, g^{\lfloor d/S \rfloor}(1)$. From these derivatives, it can reconstruct g by solving a linear system. Now, the desired value $\mathsf{DB}[i]$ is simply $g(0)$.

Woodruff and Yekhanin's scheme suffers from large upload bandwidth because they need to work with a large field size $q > d$. This is needed because otherwise, the higher-order derivatives of g may vanish to 0 and thus will not be useful for the reconstruction. Due to the simultaneously requirement $q > d$ and the requirement $\binom{m}{d} \geq n$, the query message length $\log q \cdot m$ must be super-logarithmic.

Our main idea: use Hasse derivatives rather than normal derivatives. To compress their upload bandwidth, we want to work with a small field. To achieve this, we alter their reconstruction algorithm to avoid the issue of the higher-order derivatives of g vanishing. Our main idea is to replace normal derivatives with the use of *Hasse derivatives* which do not vanish even with a small field. We show how the client can compute the Hasse derivates of g up to order $\lfloor d/S \rfloor$ denoted $\overline{\partial}^{(0)} g(1), \overline{\partial}^{(1)} g(1), \ldots, \overline{\partial}^{(\lfloor d/S \rfloor)} g(1)$ from the servers' answers using a modified version of the chain rule, and then how to reconstruct the polynomial g from the Hasse derivatives. With this change, we now only require that the field be large enough to contain an S-th root of unity, and $q = O(S)$ suffices. Further, we can choose $m = O(\log n)$ and $d = O(\log n)$ such that $\binom{m}{d} \geq n$. Now, if S is a constant, the query message length is $O(\log n)$.

Additional techniques. Using the Hasse derivatives alone can reduce the upload bandwidth to $O(\log n)$, but it is not sufficient for getting the results as tight as Theorem 1.1. To obtain these results, we need to make some further optimizations to the scheme to get rid of the requirement that the field must contain an S-root of unity. This can reduce the upload bandwidth by a constant factor — since the server space is exponentially large in the upload bandwidth, effectively we can further reduce the server space by a polynomial factor. Specifically, instead of having the client send to the servers distinct points on the line $E(i) + \lambda\vec{v}$ for a random $\vec{v} \in \mathbb{F}^q$, we instead have the client send distinct points on the curve $\lambda E(i) + \vec{v}$, by choosing $\lambda = 0, 1, \ldots, S-1$, respectively. In this case, the value $\mathsf{DB}[i]$ is equal to the coefficient of the degree-d monomial in the polynomial $g(\lambda) = F(\lambda E(i) + \vec{v})$. The server still sends back the original polynomial F as well as all derivatives up to order $\lfloor d/S \rfloor$ evaluated at the point it receives. The client still computes all the Hasse derivates of g up to order $\lfloor d/S \rfloor$ evaluated at 1. Finally, the client reconstructs the polynomial g using Hermite Interpolation, from which it can recover $\mathsf{DB}[i]$. With this modification, it suffices to set $q \geq S$ — for example, for $S = 2, 3, 4, 5$, we can simply set $q = S$.

2.2 A New Balancing Technique

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 $i \in \{0, 1, \ldots, n-1\}$ 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 $s \in \{0, 1, \ldots, S-1\}$, 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 $\{\mathsf{ans}_{r,s}\}_{s \in \{0,\ldots,S-1\}}$. 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 $n^{(2+\epsilon)/(S+1)}$.

2.3 Scalable Family of Schemes

To get a scalable family of schemes, we can also work with the same unified framework, and simply change the choice of the polynomial and parameters. Specifically, instead of using a polynomial of individual degree 1 and homogeneous total degree d, we now simply require the individual degree to be bounded by d. In other words, the total degree can be as large as $m \cdot d$. To be able to encode the entire database with such a polynomial, we need that $(d+1)^m \geq n$. We will set $S = d$ and $q = S + 1$ — for simplicity we assume q is a prime or prime power for now, and we describe the more general case in Section 7. With this parameter

choice, if we want to precompute an m-variate polynomial (or a derivative polynomial) at all possible points in $\mathbb{F}_q^m$, the total number of points $q^m = (d+1)^m = n$ is only linear in n.

The rest of the scheme follows the same framework. The client needs to reconstruct $g(\lambda) = F(E(i) + \lambda\vec{v})$ which is a degree-md polynomial in λ. For the reconstruction to be possible using Hermite Interpolation, each server needs to send back the original polynomial as well as all derivatives of order up to $md/S = m$ evaluated at the query point. The number of such derivative polynomials is upper bounded by the number of ways to choose m items from $1, \ldots, m$ with repetition, that is, $\binom{2m}{m} = O(n^{2/\log q})$. With a little more analysis, we get Theorem 1.5.

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

The scheme should satisfy the following properties:

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

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

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

4 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 A, 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 4.1. 1. Each server $s \in \{0, 1, \ldots, S-1\}$ 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 $s_1, s_2 \in \{0, \ldots, S-1\}$, 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 4.1. We also observe that these assumptions can be removed — see Appendix A for detailed proof.

4.1 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 $i \in \{0, 1, \ldots, n-1\}$ 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 $s \in \{1, \ldots, S-1\}$, 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 $s \in \{0, \ldots, S-1\}$ and stores private state $st = (st_r, b_r)$.

- **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:

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

4.2 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 $s \in \{0, 1, \ldots, S-1\}$ since rest of proof immediately follows from correctness of the underlying PIR. Since the S-server PIR scheme PIR is natural (Assumption 4.1) 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 $s = 0$,

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

4.3 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 4.1; and for $j = r$

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

4.4 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 PIR.**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 PIR.**Query** operation takes time $T_{\text{query}}(n)$ and PIR.**Recons** operation takes time $T_{\text{recons}}(n)$. The client needs to compute PIR.**Query** for every blocks but only perform PIR.**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 4.2. *Suppose there exists an S-server PIR scheme satisfying Assumption 4.1 in which achieves $C_{\text{up}}(n)$ per-server upload bandwidth, $C_{\text{down}}(n)$ download bandwidth, $T_{\text{answer}}(n)$ per-server computation, $T_{\text{query}}(n)$ **Query** operation complexity per query and $T_{\text{recons}}(n)$ **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 4.1, 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.*

5 Preliminaries on Polynomials over a Finite Field

5.1 Notations

We define $A_{k,d} \in \{0,1,\ldots,d\}^m$ to be the set of all vectors of length m and 1-norm exactly k:

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

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

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 5.1 (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 $\overline{\partial}^{(r)} f(\lambda) = \frac{1}{r!} f^{(r)}(\lambda)$.

5.2 Chain Rule

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

Lemma 5.2 (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 5.3 (Chain rule for Hasse derivatives)**.** *For m-variate polynomial $f(X_1, \ldots, X_m)$ with individual degree d over field $\mathbb{F}$, let $g(\lambda) = f(\vec{u} + \lambda\vec{v})$ be a univariate polynomial in λ, we have*

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

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 $\vec{a} \in A_{k,1}$, therefore,

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

5.3 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 5.4 (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 $e_1 + \cdots + e_m > d$. Given m distinct elements $\alpha_1, \ldots, \alpha_m \in \mathbb{F}$. 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 $\mathsf{poly}(d, \log|\mathbb{F}|)$.*

5.4 Polynomial Interpolation and Evaluation

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

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

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

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

Lemma 5.5. *For any constant $0 < \theta \le 1/2$, if we want $\binom{m}{\theta m} \ge n$ to hold, for sufficiently large n, it suffices to set $m = \frac{\log n}{H(\theta)}(1 + o(1))$, 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.*

The proof of Lemma 5.5 is deferred to Appendix C.

In [KU11, Theorem 4.1], the authors state that given an m-variate polynomial F over prime field $\mathbb{F}_q$, we can simultaneously evaluate it at every points of $\mathbb{F}_q^m$ in almost linear time:

Lemma 5.6 ([KU11])**.** *There is a deterministic algorithm that takes coefficients of an m-variate polynomial F over finite field $\mathbb{F}_q$ (W.L.O.G. 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 $O(q^m \cdot m \cdot \mathsf{poly}\log q)$.*

6 New S-Server PIR Scheme

In this section, we first propose 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, in Section 6.2, we apply the generic balancing trick of Section 4 to further reduce the bandwidth to $n^{(1+\epsilon)/(S+1)}$.

6.1 Base Construction with $n^{(1+\epsilon)/S}$ Bandwidth

We first describe a base S-server PIR scheme with Hasse derivative as building block achieving $O(n^{(1+\epsilon)/S})$ 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.

6.1.1 Construction

Parameters and notations. We will choose the following parameters.

- Let $\mathbb{F}_q$ be a finite field with order $q \geq S$, by Bertland'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 5.5.
- We use the following polynomial F over $\mathbb{F}_q$ to encode database.

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

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

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

- $\mathbf{Preproc}_s$: The same holds for each server $s \in \{0,1,\ldots,S-1\}$: for each $\vec{a} \in A_{\leq \lfloor d/S \rfloor,1}$, 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 5.6 to polynomial $\boldsymbol{\partial}^{\vec{a}} \circ F$.
- $\mathbf{Query}$: Let $i \in \{0,1,\ldots,n-1\}$ 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 $s \in \{0,\ldots,S-1\}$.
- $\mathbf{Answer}_s$: The s-th server parses the message received from the client as vector $\vec{z}_s$. For each $\vec{a} \in A_{\leq \lfloor d/S \rfloor,1}$, it sends back

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

- $\mathbf{Recons}$:

1. Define univariate polynomial $f(\lambda) = F(\vec{v} + \lambda\vec{u})$, clearly $\vec{z}_s = \vec{v} + \lambda\vec{u}$. The client computes the Hasse derivatives $\overline{\partial}^{(k)} f(\lambda_s)$ for all $s \in \{0, 1, \ldots, S-1\}$ and $0 \le k \le \lfloor d/S \rfloor$ by the chain rule (Lemma 5.3):
$$\overline{\partial}^{(k)} f(\lambda_s) = \sum_{\vec{a} \in A_{k,1}} \boldsymbol{\partial}^{\vec{a}} \circ F(\vec{z}_s) \cdot \vec{u}^{\vec{a}} = \sum_{\vec{a} \in A_{k,1}} \mathsf{ans}_{s,\vec{a}} \cdot \vec{u}^{\vec{a}} \tag{3}$$
2. Reconstruct f by its Hasse derivatives. It is obvious that f has degree at most d, for each $s \in \{0, 1, \ldots, S-1\}$, the client has already known its k-th order Hasse derivatives at point λ_s for any $0 \le k \le \lfloor d/S \rfloor$. Since $S(\lfloor d/S \rfloor + 1) > d$, the coefficients of f can be recovered by Hermite interpolation (Lemma 5.4).

   Finally, the client outputs the highest term of f, i.e., $\mathsf{Coeff}_{\lambda^d}(f(\lambda))$ as the answer.

6.1.2 Proof of Correctness

Observe that the client successfully reconstruct the polynomial f by correctness of chain rule (Lemma 5.3) and Hermite interpolation (Lemma 5.4). It remains to show that $\mathsf{Coeff}_{\lambda^d} f(\lambda)$ is exactly the answer $\mathsf{DB}[i]$. Notice that F is a polynomial of homogeneous degree d with individual degree at most 1, say

$$F(\vec{X}) = \sum_{\vec{a} \in A_{d,1}} \mathsf{Coeff}_{\vec{X}^{\vec{a}}}(F(\vec{X})) \vec{X}^{\vec{a}}$$

For each $\vec{a} \in A_{d,1}$, clearly

$$\begin{aligned} &\mathsf{Coeff}_{\lambda^d}((\vec{v} + \lambda\vec{u})^{\vec{a}}) \\ =&\mathsf{Coeff}_{\lambda^{\mathsf{wt}(\vec{a})}}((\vec{v} + \lambda\vec{u})^{\vec{a}}) \\ =&\vec{u}^{\vec{a}} \end{aligned}$$

thus

$$\begin{aligned} \mathsf{Coeff}_{\lambda^d}(f(\lambda)) &= \mathsf{Coeff}_{\lambda^d}(F(\vec{u} + \lambda\vec{v})) \\ &= \sum_{\vec{a} \in A_{d,1}} \mathsf{Coeff}_{\vec{X}^{\vec{a}}}(F(\vec{X})) \vec{X}^{\vec{a}} \cdot \mathsf{Coeff}_{\lambda^d}((\vec{v} + \lambda\vec{u})^{\vec{a}}) \\ &= \sum_{\vec{a} \in A_{d,1}} \mathsf{Coeff}_{\vec{X}^{\vec{a}}}(F(\vec{X})) \vec{u}^{\vec{a}} \\ &= F(\vec{u}) \\ &= \mathsf{DB}[i] \end{aligned}$$

6.1.3 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.

6.1.4 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 5.5) 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 $\vec{a} \in A_{\leq \lfloor d/S \rfloor,1}$, the server returns answer $\mathsf{ans}_{s,\vec{a}} \in \mathbb{F}_q$. By the fact that $|A_{\leq \lfloor d/S \rfloor,1}| = \Lambda(m, \lfloor \theta m/S \rfloor) \leq 2^{H(\theta/S)m}$ for $0 < \theta \leq 1/2$, the per-server download bandwidth is
$$O(|A_{\leq \lfloor d/S \rfloor,1}| \log S) = n^{((1+o(1))H(\theta/S)/H(\theta)} \log S.$$

- **Server computation**: For each server s and each $\vec{a} \in A_{\lfloor d/S \rfloor,1}$, 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
$$O(|A_{\leq \lfloor d/S \rfloor,1}| \log S) = n^{(1+o(1))H(\theta/S)/H(\theta)} \log S.$$

- **Client computation**: First the client computation is not less than the bandwidth i.e. $O(mS \log S + |A_{\leq \lfloor d/S \rfloor,1}| S \log S)$. 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 5.4), the time complexity of **Recons** is bounded by the arithmetic operations in $\mathbb{F}_q$ to reconstruct the Hasse derivatives of g (see Equation (3)).
For each server s and $\vec{a} \in A_{\leq \lfloor d/S \rfloor,1}$, 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
$$\begin{aligned} &O(mS \log S + |A_{\leq \lfloor d/S \rfloor,1}| \cdot Sm \log^2 S) \\ =&n^{(1+o(1))H(\theta/S)/H(\theta)} S \log^2 S. \end{aligned}$$

- **Server space**: Recall that we use a precompute-all approach: for each $\vec{a} \in A_{\leq \lfloor d/S \rfloor,1}$ and each $\vec{x} \in \mathbb{F}_q^m$, each server stores an element in $\mathbb{F}_q$. The server space is
$$\begin{aligned} &O(|A_{\leq \lfloor d/S \rfloor,1}| q^m \log S) \\ =&q^m n^{(1+o(1))H(\theta/S)/H(\theta)} \log S \\ =&n^{(1+o(1))(\log q + H(\theta/S))/H(\theta)} \\ \leq&n^{(1+o(1))(\log S + 1 + H(\theta/S))/H(\theta)} \end{aligned}$$

- **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 5.6), so the preprocessing time is bounded by $n^{(1+o(1))(\log q + H(\theta/S))/H(\theta)} \leq n^{(1+o(1))(\log S + 1 + H(\theta/S))/H(\theta)}$.

Theorem 6.1 (Our base scheme: general form)**.** *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,

$$\frac{H(\theta/S)}{H(\theta)} - 1/S \leq \frac{\ln(S)}{S(1+\ln(\frac{1}{\theta}))} \cdot (1 + O(\theta)) \tag{4}$$

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 (4) to be upper 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

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

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 C, 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 6.2 (Our base scheme: for large S)**.** *For sufficiently large S, n, and sufficiently small $\epsilon > 0$, there exists an S-server PIR scheme such that, which achieves $O(n^{(1+\epsilon)/S} \log S)$ per-server bandwidth, $O(n^{(1+\epsilon)/S} \log S)$ per-server computation and $O(n^{(1+\epsilon)/S} S \log^2 S)$ client computation per query, with $n^{0.368\epsilon S^{(1+o(1))/\epsilon}}$ preprocessing time and server storage.*

Remark 6.3. It is not hard to see that the above Corollary 6.2 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 3 compares the exact exponents of server storage for some concrete server numbers when $\epsilon = 0.5$.

Table 3: **Numerical Experiments for** $\epsilon = 0.5$. The last three columns represents the exponents of communication/work, server storage of our base scheme 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**
2	2	0.4110	0.75	1.7735	1.7735
3	3	0.2259	0.5	2.5563	3.0947
4	4	0.1410	0.375	3.7832	5.4874
5	5	0.0956	0.3	5.4051	9.0947
6	7	0.0687	0.25	8.0230	14.0940
7	7	0.0516	0.2143	9.7838	20.0667
8	8	0.0402	0.1875	12.5322	28.9918
9	9	0.0321	0.1667	15.6510	39.2448
10	11	0.0262	0.15	19.9255	51.5976

6.2 Applying the Balancing Technique to Reduce Bandwidth

Combining our base S-server PIR scheme with balancing technique (Lemma 4.2), we immediately obtain the desired result:

Corollary 6.4. *For any $\epsilon \in (0,1)$ and $0 < \theta \le 1/2$, 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, and

$$O(n^{S\alpha(H(\theta/S)/H(\theta)+o(1))} S \log^2 S)$$

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 6.5. *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 Scalable Family of Schemes

In this section, we show how to use the same unified framework to get a scalable family of schemes, by changing the choice of the polynomial and the parameters.

7.1 Additional Preliminaries: Fast Polynomial Interpolation

In Lin et al. [LMW23], authors present an interpolation algorithm [LMW23, Lemma 2.2], in order to encode a database of at most q^m elements into a m-variate polynomial F over $\mathbb{F}_q$:

- Given an **injective** map $E : \{0, 1, \ldots, n-1\} \to \mathbb{F}_q^m$, there is an interpolation algorithm that takes $n \leq q^m$ values $\{y_i\}_{i \in \{0,1,\ldots,n-1\}}$, and recovers coefficients of a polynomial $F(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$ with individual degree $q-1$ in each variable such that $F(E(i)) = y_i$ for all $i \in \{0, 1, \ldots, n-1\}$. Further, the algorithm runs in time $O(q^m \cdot m \cdot \mathsf{poly} \log q)$.

7.2 Construction

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

- Let $S(n)$ be 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 $q = S^*(n) + 1$, and will work on finite field $\mathbb{F}_q$.
- Set $m = \lceil \log n / \log q \rceil$ such such that $q^m \geq n$.

S-server PIR. Our S-server PIR works as follows. It is analogous to scheme in Section 6.1 except that we flip the roles of query vector $\vec{u}$ and randomized vector $\vec{v}$ so that we can minimize the degree of encoded polynomial F.

- $\mathbf{Preproc}_s$: Encode database DB to m-variate polynomial F with individual degree $d = q - 1$. Concretely, we construct $E : \{0, 1, \ldots, n-1\} \to \mathbb{F}_q^m$ be an *injective* index function, and recover F by interpolating on the set $\{\mathsf{DB}[i]\}_{i \in \{0,1,\ldots,n-1\}}$ using the techniques described by Lin et al. [LMW23].

  So for each $\vec{a} \in A_{\leq m,d}$, we can use the preprocessing algorithm described in Lemma 5.6 to precompute 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 $S^*(n)$ distinct and **nonzero** elements in $\mathbb{F}_q$ called $\lambda_0, \ldots, \lambda_{S^*(n)-1}$.

  For $s \in \{0, 1, \ldots, S^*(n) - 1\}$, the client sets

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

  The client sends $Q_s = \vec{z}_s$ to each server $s \in \{0, \ldots, S^*(n) - 1\}$.

- $\mathbf{Answer}_s$: The s-th server ($s \in \{0, 1, \ldots, S^*(n)\}$) parses the message received from the client as a vector $\vec{z}_s$. For each $\vec{a} \in A_{\leq m,d}$, 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 $\vec{z}_s = \vec{u} + \lambda_s \vec{v}$. Given the responses of all servers, the client computes $\overline{\partial}^{(k)} f(\lambda_s)$ for all $s \in \{0, 1, \ldots, S-1\}$ and $0 \leq k \leq m$ by:

  $$\begin{aligned} \overline{\partial}^{(k)} f(\lambda_s) &= \sum_{\vec{a} \in A_{k,d}} \overline{\partial}^{\vec{a}} \circ F(\vec{z}_s) \cdot \vec{v}^{\vec{a}} \\ &= \sum_{\vec{a} \in A_{k,d}} \mathsf{ans}_{s,\vec{a}} \cdot \vec{v}^{\vec{a}} \end{aligned}$$

  2. Reconstruct f by its Hasse derivatives. Since F has individual degree $d = q - 1$, f has degree $\leq m \cdot (q-1)$. For each $s \in \{0, 1, \ldots, S-1\}$, the client has already known its k-th order Hasse derivatives at point λ_s for any $0 \leq k \leq m$, thus in total it has $(m+1)S^*(n) = (m+1)(q-1) > \deg(f)$ information about f. So the client can use Hermite interpolation (Lemma 5.4) to reconstruct the coefficients, and output $f(0) = F(\vec{u}) = \mathsf{DB}[i]$ as the answer.

7.3 Proof of Correctness

The correctness of PIR scheme immediately follows from the correctness of chain rule (Lemma 5.3) and Hermite interpolation (Lemma 5.4).

7.4 Proof of Security

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

7.5 Efficiency

We now analyze the efficiency of our construction.

- **Bandwidth**: The bandwidth will be dominated by the download bandwidth. Each server should return $|A_{\leq m,d}|$ elements in $\mathbb{F}_q$, and we have

$$|A_{\leq m,d}| \leq \binom{2m}{m} \leq 2^{2m} = 4^m = O(n^{2/\log q}),$$

  Thus the total per-server bandwidth is bounded by $|A_{\leq m,d}| \log q = O(n^{2/\log q} \log q)$.

- **Server computation**: Each server simply sends back $|A_{\leq m,d}| = O(n^{2/\log q})$ stored values, so the total computation is same as bandwidth, that is, $O(n^{2/\log q} \log q)$.

- **Client computation**: The client computation is bounded by the total bandwidth, i.e. $O(S^*(n) \cdot |A_{\leq m,d}| \log q)$. Recall that $S^*(n) \leq S(n)$ and $|A_{\leq m,d}| = O(n^{2/\log q})$, so client computation is $O(n^{2/\log q} \cdot S(n) \log q)$.

- **Server space**: Each server should do the preprocessing algorithm in Lemma 5.6 to $|A_{\leq m,d}| = O(n^{2/\log q})$ polynomials. For each polynomial, it needs to store q^m elements in $\mathbb{F}_q$, thus in total it takes space

$$\begin{aligned} &|A_{\leq m,d}| \cdot O(q^m \log q) \\ =&O(n^{2/\log q}) \cdot O(n \log^2 q) \\ =&O(n^{1+2/\log q} \log^2 q). \end{aligned}$$

  where the first equation is due to the fact $q^m \leq nq$.

- **Preprocessing time**: Each server should preprocess $|A_{\leq m,d}| = O(n^{2/\log q})$ polynomials. It follows from Lemma 5.6, that this takes preprocessing time

$$\begin{aligned} &|A_{\leq m,d}| \cdot O(q^m \cdot m \cdot \mathsf{poly} \log q) \\ =&O(n^{2/\log q}) \cdot O(n \cdot \mathsf{poly}(m, \log q)) \\ =&n^{1+2/\log q} \cdot \mathsf{poly} \log n. \end{aligned}$$

Notice that our parameterization guarantees $q > S(n)/2$ and $2/\log q \leq 2/(\log S(n) - 1)$, therefore we have:

Theorem 7.1. *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 $O(n^{2/(\log S-1)} S \log S)$ client computation per query, with $n^{1+2/(\log S-1)} \cdot \mathsf{poly} \log n$ preprocessing time and server storage.*

For constant number servers setting, our construction shows nontrivial results about tradeoff between preprocessing time and bandwidth. In comparison to Corollary 6.2, here we significantly save preprocessing time and server storage while require larger bandwidth and computation per query.

Moreover, 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 $|A_{\leq m,d}|$ is bounded by $n^{o(1)}$. Moreover, the polylogarithm factor can be absorbed by $n^{o(1)}$. In conclusion, we have:

Corollary 7.2. *For any $S(n) = \omega(1)$, there exists an $S(n)$-server PIR scheme such that, it can achieve $n^{o(1)}$ per-server communication, $S(n) \cdot n^{o(1)}$ per-server computation and $n^{o(1)}$ client computation per query, with $n^{1+o(1)}$ preprocessing time and server storage. Specifically, when $S(n) = n^{o(1)}$, the client computation is also bounded by $n^{o(1)}$.*

Acknowledgments

We gratefully acknowledge Yuval Ishai and Henry Corrigan-Gibbs for suggesting the generic balancing technique. The version described in our paper is a slight improvement of their original idea. This work is in part supported by NSF awards 2128519 and 2044679, an ONR grant, and a DARPA SIEVE grant under a subcontract from SRI.

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.

[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.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In *TCC*, 2017.

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

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

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

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

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

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In *TCC*, 2017.

[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.

[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/.

[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.

[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 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 A.1, we show how to transform any PIR scheme to one that satisfies the natural assumptions with an S factor blowup. Next, in Appendix A.2, we show that we can save the S-factor blowup by not going through the "arbitrary to natural" transformation of Appendix A.1. In fact, we can improve our balancing technique of Section 4 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.

A.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 4.1, with S factor blowup.

A.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 $s \in \{0, 1, \ldots, S-1\}$: for each $s' \in \{0, 1, \ldots, S-1\}$, invoke $\widetilde{\mathsf{DB}}_{s'} = \mathsf{PIR}.\mathbf{Preproc}_{s'}(\mathsf{DB})$ and stores the preproccesing result $\{\widetilde{\mathsf{DB}}_{s'}\}_{s' \in \{0,1,\ldots,S-1\}}$.
- $\mathbf{Query}$: 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 $(Q_{(s'-s) \bmod S, s'})_{s' \in \{0,1,\ldots,S-1\}}$ 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 $s' \in \{0, 1, \ldots, S-1\}$, it simulates the behavior of s'-th server of the underlying PIR scheme PIR taking message $Q'_{s',s}$ as input.

  Formally, for all $s' \in \{0, 1, \ldots, S-1\}$ 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 $\mathsf{DB}[i]$:

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

A.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.

A.1.3 Proof of Security

We first prove the sematical security of our PIR scheme: each server $s \in \{0, 1, \ldots, S-1\}$ 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:

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

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 4.1.

A.1.4 Efficiency

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

Lemma A.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 4.1 with S factor blowup of bandwidth, efficiency and server storage.*

Combining it with Lemma 4.2, we obtain:

Corollary A.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 4.1, 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.*

A.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):

A.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 4 and Appendix A.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 $i \in \{0, 1, \ldots, n-1\}$ be the queried index, and $r = \lfloor i/n^\mu \rfloor$ be the block where i resides.

  For block $j = r$, client indepently 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

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

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 $s \in \{0, \ldots, S-1\}$ 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 4:

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

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

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

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

A.2.2 Proof of Correctness

The correctness proof is essentially same as Section 4.2, we omit here.

A.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 4.3.

A.2.4 Efficiency

Comparing to scheme in Section 4, 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 A.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 A.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 4.1 from Lemma 4.2 while remains the asymptotic result.

B The Case of Polylogarithmically Many Servers

For the special case of polylogarimically many servers, 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.

B.1 Construction

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

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

- $\mathbf{Preproc}_s$: Encode database DB to m-variate polynomial F with individual degree $d = q - 1$. Concretely, we construct $E : \{0, 1, \ldots, n-1\} \to \mathbb{F}_q^m$ be an *injective* index function, and recover F by interpolating on the set $\{\mathsf{DB}[i]\}_{i \in \{0,1,\ldots,n-1\}}$ 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 5.6.

  Moreover, each server s should 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

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

  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 $s \in \{0, 1, \ldots, S-1\}$, the client sets

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

  The client sends $Q_s = \vec{z}_s$ to each server $s \in \{0, \ldots, S-1\}$.

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

  $$\mathsf{ans}_s = \underbrace{F(\vec{z}_s)}_{\text{precomputed during preproc}} \cdot w_s$$

  to the client.

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

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

B.2 Proof of Correctness

The correctness of PIR scheme just follows from a standard Lagrange interpolation.

B.3 Proof of Security

The security proof is same as Section 7.4.

B.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 $O(m \log q) = O(\log n/\epsilon)$.
- **Server computation**: Since both $F(\vec{z}_s)$ and w_s are precomputed, the server computation is bounded by total bandwidth, that is, $O(\log n/\epsilon)$.
- **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 $O(Sm \log q + S \log q) = O(\log^{2+\epsilon} n/(\epsilon^2 \log\log n))$.
- **Server space**: Each server should store $F(\vec{x})$ for all $\vec{x} \in \mathbb{F}_q^m$. There are q^m elements in $\mathbb{F}_q$, thus in total it takes space

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

  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 5.6, it takes time

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

  Since $1/(\epsilon \log\log n)$ goes to 0 as n goes to infinity, we have

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

C Proof of Lemma 5.5

We now prove Lemma 5.5. By Stirling's approximation, we have

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

Since $H(\theta) \geq \theta(1-\theta)$ for $\theta \in [0,1]$, the above is lower bounded by $2^{H(\theta)m-0.5\log(H(\theta m))-O(1)} \geq 2^{H(\theta)m(1-o(1))}$. 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.