Multi-Server Doubly Efficient PIR

Arthur Lazzaretti Zeyu Liu Ben Fisch Charalampos Papamanthou [*]

Yale University

Abstract

Doubly Efficient Private Information Retrieval (DEPIR) pertains to a Private Information Retrieval (PIR) scheme with both near-linear server-side preprocessing time and sublinear query time. Very recently, Lin et al. (STOC '23) devised the first single-server DEPIR scheme from standard assumptions. However, their work left one major question open: can we build a DEPIR scheme in the multi-server setting, without relying on heavy cryptographic machinery?

In this work, we propose the first doubly efficient information-theoretic PIR scheme, in the multi-server setting. For a database of size N, our scheme allows servers to answer an infinite number of client queries in $N^{o(1)}$ time, after a single preprocessing phase which takes $N^{1+o(1)}$ time. Our result is only a $N^{o(1)}$ factor from the lower bound proven by Persiano and Yeo (SODA '22) for this setting.

In addition, we devise a second information-theoretic PIR scheme which pushes the state of the art for the setting where the number of servers is more constrained. It achieves equally fast query times as our first scheme above, and a preprocessing time of $N^{2+o(1)}$.

1 Introduction

Private Information Retrieval (PIR) was first introduced almost three decades ago by Chor et al. [15] in the multi-server, information-theoretic setting. In this problem, a client holds an index $i \in [N]$, and k, non-colluding servers hold the same database $\mathsf{DB} \in \{0,1\}^N$. The goal of the PIR protocol (informally) is to devise a way for the client to learn $\mathsf{DB}[i]$ without either server learning any information about the client's desired index i.

Since the problem's original inception, many works have pushed our understanding of the PIR problem in this same setting [14, 14, 6, 39, 41] (a non-exhaustive list) as well as many different scenarios; single-server computational PIR [29], two-server computational PIR [22, 12] among others [8, 42, 9]. Aside from the number of servers and the computational assumption on the adversary, we can also consider preprocessing or non-preprocessing schemes. Within preprocessing schemes, there are two well-known approaches, which we will denote server preprocessing and client preprocessing.

In client preprocessing, the server and client run a joint preprocessing phase after which the client stores a secret hint which can be used to perform queries to the server later in time sublinear in the database size. This requires each client to run an expensive (linear time) preprocessing phase with the server, which means the server work to server m clients is at least $m \cdot N$, where N is the database size.

[*]Authors are alphabetically ordered students followed by advisors.

On the other hand, in server preprocessing, the server runs a one-time preprocessing phase by itself and stores additional bits server-side, which it can then use to answer queries from any client in time sublinear in the database size. In this work, we focus on the server preprocessing setting (we cover a broad overview of the other settings in Section 1.2).

The server preprocessing phase has a clear advantage with respect to total server time, in the sense that the overhead for serving many clients is considerably smaller. The first family of server preprocessing schemes was introduced by Beimel et al. [6] in 2000 in the multi-server setting. They present schemes for constant and polylogarithmic number of servers, with different trade-offs, as well as a number of lower bounds. Notably, in its Journal of Cryptology version [7], the authors present the first multi-server scheme (with a polylogarithmic number of servers) in the server preprocessing model which achieves $\mathsf{polylog}(N)$ query time and $\mathsf{polylog}(N)$ communication after a preprocessing phase running in time $\mathsf{poly}(N)$

Persiano and Yeo [36] recently tighten the lower bounds shown by Beimel et al., showing that any server preprocessing scheme in the single server setting with query time $\Omega(t)$ must have the server store a hint of at least size $\Omega(N \log N/t)$, for any number of servers.

Very recently, in a seminal paper by Lin et al. [31] which makes use of a polynomial preprocessing algorithm by Kedlaya and Umans [27] to achieve a PIR scheme which preprocesses in time $O_\lambda(N^{1+o(1)})$ and stores $O_\lambda(N^{1+o(1)})$ after which it can serve PIR queries in time $N^{o(1)}$.[1] They refer this as the *doubly efficient PIR*: in particular, the server can preprocess the database *once in nearly linear time* (e.g., $N^{1+o(1)}$), and answer an infinite amount of queries from the clients very quickly (e.g., $N^{o(1)}$ time).

According to the bound by Persiano and Yeo, this is close to optimal asymptotically (only $N^{o(1)}$ factors away). After the work by Lin et al., two main questions were left open for server preprocessing PIR:

1. Can we achieve similar asymptotics in the multi-server setting without the computational assumption (i.e., information-theoretic) while minimizing the number of servers required?

2. Can we achieve doubly efficient PIR without the use heavy cryptographic primitives such as fully homomorphic encryption (FHE)?

The first question is important theoretically due to the relationship of multi-server information-theoretic PIR with other cryptographic primitives [41, 40] and to push our understanding of the problem. The second question was also posed after a follow-up work by Okada et al. [35] showed that the scheme by Lin et al., implemented as is, still incurs astronomic empirical costs despite the very nice asymptotic efficiencies.

In this work, we answer both questions in the affirmative by providing a novel, multi-server server preprocessing scheme with near-optimal asymptotics that does not use FHE. Furthermore, since our constructions do not rely on heavy cryptographic machinery (e.g., FHE), we believe they may pave the way towards building a truly practical doubly efficient PIR constructions.

1.1 Our Results

In this paper, we construct two k-server information-theoretic PIR construction with server-side preprocessing. Both are compared with the state-of-the-art prior works in information-theoretic PIR with server-side preprocessing [7, 39] and the concurrent work [21] in Table 1.

[1]Or alternatively preprocess in $N^{1+\epsilon}$ time and query in $\mathsf{polylog}(N)$ time for some constant ϵ).

Scheme	Efficiency			
	# of Servers	Preprocessing Time & Storage	Online Time	Communication
[39]	2	$\mathsf{poly}(N)$	$N/\mathsf{polylog}(N)$	$N^{1/3}$
[7]	k	$\mathsf{poly}(N)$	$O(N^{1/k+\epsilon})$	$O(N^{1/k+\epsilon})$
[21]	k	$\mathsf{poly}(N)$	$O(N^{1-\gamma(k-1)+\epsilon})$	$O(N^{\gamma+\epsilon})$
[7, 21]	$O(\mathsf{polylog}(N))$	$\mathsf{poly}(N)$	$\mathsf{polylog}N$	$\mathsf{polylog}N$
Theorem 4.1	$\widetilde{O}(\log(N))$	$N^{1+o(1)}$	$N^{o(1)}$	$N^{o(1)}$
Theorem 5.2	$o(\log(N))$	$N^{2+o(1)}$	$N^{o(1)}$	$N^{o(1)}$

Table 1: Comparisons to prior works. N is the number of bits in a database, and k, ϵ are tunable constants in [7]. γ is a tunable constant satisfying $\frac{1}{k+1} \leq \gamma \leq \frac{1}{k}$ in [21]. Our $\widetilde{O}(\log(N)) = \frac{\log(N)\log\log(N)}{\log\log\log(N)}$.

In particular, our *first* scheme is the first doubly efficient PIR scheme (see Theorem 2.1) with no computational assumptions where the server computation at query time is significantly smaller than N, specifically, $N^{o(1)}$. Also, in order to satisfy the definition of doubly efficient, it means that each server's preprocessing time is near-linear in the database size N (in particular, we have $N^{1+o(1)}$). These are achieved with roughly $O(\log(N))$ servers.

In our second scheme, we demonstrate that one can further reduce the number of servers required for the scheme, at the expensive of a more costly preprocessing phase, which is no longer near linear, but $N^{2+o(1)}$. This is the first scheme to achieve $N^{o(1)}$ query time with $\mathsf{poly}(N)$ preprocessing and $o(\log N)$ servers. Note that such efficiency is also not achieved by any prior work with $o(\log(N))$ servers (even *any* $\mathsf{poly}(N)$ preprocessing time).

To transfer the doubly efficient PIR scheme of Lin et al. [31] into the information-theoretic setting, the first challenge is to find a way to hide the query without employing homomorphic encryption. In the multi-server setting, secret sharing has been a solution to hide the client query without the use of encryption [37, 4]. However, it is not immediately clear how to secret share the input in a way that preserves the homomorphism necessary to evaluate the polynomial from the server side without blowing up the cost.

We therefore take a slightly different approach, inspired by the elegant result of Woodruff and Yekhanin [39], which removes the necessity for finding a secret sharing scheme with homomorphic properties. In particular, they ask each server to evaluate a single point on the polynomial representing the database, and the client can recover the database point using these evaluations. Following this work, we can apply the polynomial-preprocessing data structure considered in [31]. In more detail, we can let each server preprocess the polynomial in advance, and then evaluate the preprocessed result online.

One immediate issue, however, is that using the result in [31], the resulting scheme has relatively bad efficiency. In particular, we need the preprocessing time to be $\mathsf{poly}(N)$ or we need $\mathsf{polylog}(N)$ servers with some large hidden factor in the exponent (i.e., $\log^r(N)$ for some large r, i.e. $r > 20$).

Thus, instead, We also employ a different polynomial preprocessing technique [11] than the one used in [31]. Such new result introduces some new constraints (e.g., the characteristic of $\mathbb{F}_q$ the polynomial works over p needs to be $d^{o(1)}$ where d is the individual degree of the polynomial), and thus we need to carefully balance how the parameters are chosen to make sure the preprocessing works and achieve the desired efficiency. We expand on these in more detail in Sections 4 and 5.

Comparison with prior information-theoretic PIR works. As shown in Table 1, compared to [7] with constant k number of servers, our construction achieves better efficiency (both for preprocessing and online queries). A similar comparison can be drawn when compared to [39]. Compared to [7] with polylogarithmic number of servers, our construction achieves better preprocessing efficiency, while requiring a smaller number of servers. On the other hand, our online efficiency is slightly worse. Nonetheless, altogether, our work provides a preprocessing time and query time trade off not achievable before.

Comparison with concurrent and independent work. A very recent work [21] also focuses on information-theoretic multi-server PIR with server-side preprocessing.

In terms of results, in [21, Thm 1.3 & Appendix B], they achieve $\mathsf{poly}(N)$ preprocessing time and storage, $\mathsf{polylog}(N)$ online costs (computation and communication) with $\mathsf{polylog}(N)$ servers, which asymptotically is similar to [7, Thm 4.9] (summarized in Table 1, the fourth row).

Compared to our result, our result achieves better preprocessing time than both schemes (with fewer servers), as seen in Table 1 (although our query time is slightly worse, it is not clear that either scheme can achieve our preprocessing time, even at the sacrifice of their query time).

The other results in [21] provide alternative constructions for a constant number of servers which allow finer tuning than prior works in terms of online computation and communication (see Table 1, the third row): allowing better communication by trading-off some online computation cost. On the other hand, our paper focuses on providing a tradeoff between the number of servers and the preprocessing time, for a reasonably small number of servers (roughly or less than $\log(N)$ servers), while still maintaining small online costs (i.e., $N^{o(1)}$).

In terms of techniques, our works share similarities as both are based on [39] and both use polynomial preprocessing. However, they use Hasse derivatives while we use normal derivatives, as well as different techniques and a different polynomial preprocessing scheme.

1.2 Other Related Works

Client preprocessing PIR was first introduced in the two-server setting by Corrigan-Gibbs and Kogan [18, 28]. As aforementioned, in this setting, the client and server jointly run a preprocessing phase and the client stores a hint which it later uses to perform queries in time sublinear in the database size. Although the server has to run one preprocessing phase per client, one big advantage of this model is that the server stores no additional bits server-side other than the original database. In contrast, server preprocessing schemes can require quite large server storage overheads, up to 100x the size of the original database [6, 31].The original work also showed a lower bound on the efficiency of schemes in client preprocessing model. In specific, if the preprocessing phase outputs a hint of size $O(Q)$, then the query phase must at least in time $O(N/Q)$.Several works have since improved the original work by Corrigan-Gibbs et al. in terms of bandwidth [38], as well as devised single-server client preprocessing schemes with similar techniques and asymptotics [17, 30, 43, 19].

One line of work in server preprocessing not mentioned above, by [13, 26] achieved PIR schemes with $\widetilde{O}(\mathsf{poly}(N))$ preprocessing time and $\mathsf{polylog}(N)$ both client and server time per query from non-standard assumptions, which have yet to see a formal security reduction to known hard problems.

Other works have studied preprocessing that does not reduce asymptotic server load but performs extremely fast in practice in the single server setting [24, 32, 33, 1]

PIR has been used as a building block for many different privacy-preserving applications, such as meta-data hiding messaging [2], private streaming [23], private contact-tracing [25], and more [3, 32].

2 Preliminary

In this section we will cover important primitives and definitions necessary for our work.

Notation. $\mathbb{F}_q$ defines a finite field of size q. The characteristic of a finite field is the smallest positive number of copies of the field's multiplicative identity (i.e., 1) that will sum to the additive identity (i.e., 0). If q is a prime power (i.e., $q = p^\alpha$ for some prime p and $\alpha \geq 0$), the characteristic of $\mathbb{F}_q$ is p. We use $\widetilde{O}(\log(N))$ to represent $O(\log(N)\mathsf{poly}\log\log(N))$.

2.1 Private Information Retrieval

First, we define PIR formally. This definition captures the informal requirements of our client always being able to reconstruct the desired database entry at the end of the protocol, as well as the requirement that servers, individually, not learn anything about the query index picked by the client.

Definition 2.1 (Multi-server Doubly Efficient Private Information Retrieval with server preprocessing [15, 31])**.** A k-server, server preprocessing Private Information Retrieval scheme for a database of length N consists of four PPT algorithms:

- A preprocessing function $\mathsf{Preprocess}$: $\mathsf{DB} \in \{0,1\}^N \to \widetilde{\mathsf{DB}}$;
- a query function $\mathsf{Query} : (i \in [N], j \in [k]) \to q$;
- an answer functions $\mathsf{Answer} : (\widetilde{\mathsf{DB}}, q) \to A$;
- a reconstruction function $\mathsf{Reconstruct} : A_1, \ldots, A_k \to \{0,1\}$.

These functions should satisfy the correctness and privacy defined below.
Correctness: For every $\mathsf{DB} \in \{0,1\}^N$, let $\widetilde{\mathsf{DB}} = \mathsf{Preprocess}(\mathsf{DB})$. For any $i \in [N]$, let $q_j \leftarrow \mathsf{Query}(i, j)$, and $A_j \leftarrow \mathsf{Answer}(\widetilde{\mathsf{DB}}, q_j)$, where $j \in [k]$, it holds that

$$\Pr[\mathsf{Reconstruct}(i, A_1, \ldots, A_j) = \mathsf{DB}[i]] = 1$$

Privacy: For every i_1,$i_2 \in [N], j \in [k]$, and $q \in \mathbf{Q}$ where $\mathbf{Q}$ is the output space of Query,

$$\Pr[\mathsf{Query}(i_1, j) = q] = \Pr[\mathsf{Query}(i_2,j) = q]$$

Then, a PIR scheme is doubly-efficient if it satisfies the following property.
Doubly efficient: A PIR construction is *doubly efficient* if it is both *Preprocessing efficient* and *Query efficient*, which are defined as follows:

- (Preprocessing efficient) For any database $\mathsf{DB} \in \{0,1\}^N$, the runtime of $\widetilde{\mathsf{DB}} \leftarrow \mathsf{Preprocess}(\mathsf{DB})$ and $|\widetilde{\mathsf{DB}}|$ are both $N^{1+o(1)}$.

- (Query efficient) For any $i \in [N]$, let $q_j \leftarrow \mathsf{Query}(i,j)$, $A_j \leftarrow \mathsf{Answer}(\widetilde{\mathsf{DB}}, q_j)$, for all $j \in [k]$, and $\mathsf{Reconstruct}(i, A_1, \ldots, A_j)$, it holds that:
    1. The total runtime of the k calls of $\mathsf{Query}, \mathsf{Answer}$ and the one call of $\mathsf{Reconstruct}$ in total are $N^{o(1)}$.
    2. $|(q_1, \ldots, q_k, A_1, \ldots, A_k)| = N^{o(1)}$.

Note that this definition naturally captures the multi-query setting since it assumes correctness for any query after the preprocessing. This definition is slightly modified from the original definition presented in [15] (and adapted according to [31] to accommodate the doubly efficient definition).

2.2 Polynomial preprocessing

Polynomial preprocessing consists of two steps. The first step, preprocessing, takes in a polynomial and outputs a datastructure. Subsequently, this data structure can be used to evaluate the polynomial at any point in time much faster than the naive evaluation algorithm using only the polynomial.

We restate the results from Bhargava et al. [11] as follows.[2]

Theorem 2.2 (Thm 6.1 in [11])**.** *For a m-variate polynomial $P : \mathbb{F}_q^m \to \mathbb{F}_q$ with individual degree $\leq d$, over some finite field $\mathbb{F}_q$ such that $q = p^\alpha$ for some prime $p = d^{o(1)}$ and $\alpha > 0$ then, there exists an algorithm* $\mathsf{PolyPreProcess}$ *such that (1) the runtime of* $\mathsf{PolyPreProcess}(P)$ *is $(\alpha dp)^m \cdot 4^m \cdot \mathsf{poly}(\alpha, d, m, p)$; (2) let $\widetilde{P} \leftarrow \mathsf{PolyPreProcess}(P)$, for any $\vec{x} \in \mathbb{F}_q^m$, $\widetilde{P}(\vec{x}) = P(\vec{x})$ and that the runtime of $\widetilde{P}(\vec{x})$ is $4^m \cdot \mathsf{poly}(\alpha, d, m, p)$.*

With appropriate parameter choices, we will be able to find a polynomial P that represents our database of N bits such that we can preprocess P into a data structure in time $O(N^{1+o(1)})$ and subsequently evaluate P at any point in time $O(N^{o(1)})$. We dive more into the exact details in Section 4.

2.3 Polynomial Interpolation

Notice that to be able to use the polynomial preprocessing scheme mentioned above in a PIR scheme, we need to find a polynomial that represents the database. For any $x \in \{0,1\}^N$, we need to find a polynomial P such that for any $i \in [N]$, $P(\mathsf{encode}(i)) = x_i$, for some encoding function encode to be defined later. This is the problem of polynomial interpolation. Given a set of N points, we need to fit a polynomial to these points. It is well-known that we can find a univariate polynomial of degree N that interpolates N points in time $O(N \log N)$ [31, 20].This result is also extended to the multi-variate polynomial setting, as seen in the theorem below. [3]

Lemma 2.3 (Lemma B.1 in [31])**.** *Let $\mathbb{F}_q$ be a finite field, and let $m \in \mathbb{N}$ be an integer. Let $\{y_{x_1,\ldots,x_m} \in \mathbb{F}_q\}_{(x_1,\ldots,x_m) \in \mathbb{F}_d^m}$ be any set of d^m values. Then, there is an algorithm that runs in time*

[2]This is a follow-up work to [27] which is used by [31] in their PIR construction, with slightly different asymptotics and tradeoffs. Note that [10] improves the result from [11] on some fronts, but some technical details make it unusable for our use case.

[3]Note that in [31], their lemma is stated in a prime field. However, it easily extends to any finite field, as the uni-variate polynomial interpolation used in its proof extending to multi-variate interpolation can be done in any finite field as shown in [20].

$O(d^m \cdot m \cdot \mathsf{polylog}(q))$ and recoveres the coefficients of a polynomial $f(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$ with individual degree less than d in each variable such that $f(x_1, \ldots, x_m) = y_{x_1,\ldots,m_m}$ for all $(x_1, \ldots, x_m) \in \mathbb{F}_q^m$.

3 Multi-server PIR by Woodruff and Yekhanin [39]

We first recall in detail the scheme by Woodruff and Yekhanin [39] in the k-server, no preprocessing PIR setting, which elegantly achieves large bandwidth improvements with respect to its predecessors. Our construction greatly relies on this work.

Let $h = 2k-1$ where recall k is the number of non-colluding servers available in the PIR scheme. We first define an encoding function used in [39] $\mathsf{encode} : [N] \to \mathbb{F}_p^m$ where $m = O(hN^{1/h})$ and $k < p < 2k$. We define $\mathsf{encode}(1), \ldots, \mathsf{encode}(N)$ to be distinct points of hamming weight h in $\mathbb{F}_p^m$ with each individual value restricted to $\{0,1\} \subset \mathbb{F}_p$. Such encoding exists if C(m,h) $\geq N$, where C here denotes the number of combinations [5, 39]. (This is database-independent but depends on public parameters such as the size of the database and the number of servers.)

Now, suppose the servers also are given along with the database an m-variate polynomial F such that for each $i \in [N]$, $F(\mathsf{encode}(i)) = \mathsf{DB}[i]$ (this can be interpolated in near-linear time).

Note F is a m-variate polynomial with a total degree of $2k-1$ which maps $\mathbb{F}_p^m \to \mathbb{F}_p$. Explicitly, F can be written as:

$$F(z_1, \ldots, z_m) = \sum_{i=1}^{N} \left(\mathsf{DB}[i] \cdot \prod_{\mathsf{encode}(i)[\ell]=1} z_\ell \right),$$

where $\mathsf{encode}(i)[\ell]$ denotes the $\ell-$th element of the output of $\mathsf{encode}(i)$. One can work out that $F(\mathsf{encode}(i)) = \mathsf{DB}[i]$. Also, since each output of encode has weight h, it follows that also F has total degree at most h (at most h variable are multiplied together per term). From this setup, the client performs a query to an index $i \in [N]$ as follows:

1. Pick $\vec{v}$ uniformly at random from $\mathbb{F}_p^m$.
2. To each server $j \in [k]$, send $q_j = \mathsf{encode}(i + \lambda_j \vec{v})$, where each λ_j is a distinct non-zero element from $\mathbb{F}_q$.
3. Server j computes and returns $F(q_j)$ as well as $\frac{\partial F}{\partial z_1}(q_j), \ldots, \frac{\partial F}{\partial z_m}(q_j)$.

The last part of item 3 refers to the partial derivative of F with respect to variable z_i evaluated at point q_j, for each $i \in [m]$.

First, note that we can define a function $f : \mathbb{F}_q \to \mathbb{F}_q$ such that $f(\lambda_j) = F(i + \lambda_j \vec{v})$. This function has a degree at most $h = 2k-1$ because F has total degree $h = 2k-1$ as explained above. Also note that $f(0) = F(i) = \mathsf{DB}[i]$ by construction.

If it were the case that $h = k-1$, we could recover the entire function f (and thus $f(0) = \mathsf{DB}[i]$) with each server returning only $F(\mathsf{encode}(i) + \lambda_j \vec{v}) = f(\lambda_j)$ and then interpolating. However, since $h = 2k-1$, the max degree of our local polynomial f is at most $2k-1$ and therefore k points are not enough to recover $f(0)$. The observation by [39] is that we can use the fact that to interpolate a polynomial of degree $2k-1$, it suffices to know k points of the polynomial and the derivative at these same k points. Now, from the partial derivatives returned by each server, we can recover $f'(\lambda_j)$ by applying the chain rule as follows:

$$f'(\lambda_j) = \frac{\partial F(i + \lambda_j \vec{v})}{\partial \lambda} = \sum_{\ell=1}^{m} \vec{v}_\ell \cdot \frac{\partial F}{\partial z_\ell}(\mathsf{encode}(i) + \lambda_j \vec{v}).$$

Since the client knows the first element in the sum and the server returns the second part, it can therefore compute $f'(\lambda_j)$ for each $j \in [k]$ and use these along with the set of $\{f(\lambda_j)\}_{j\in[k]}$ to compute $f(0) = F(\mathsf{encode}(i) + 0 \cdot \vec{v}) = \mathsf{DB}[i]$.

With this, the client can recover $f(0) = \mathsf{DB}[i]$ the data point with k servers and $O(k^2 \log k N^{1/(2k-1)})$ of communication cost.

This scheme is the starting point for our multi-server scheme with preprocessing. Note that [39] do include mention of preprocessing PIR in their work as well, but the best they can achieve is a query time of $O(N/\log^r N)$ for some constant r. In contrast, we will pair this technique with the polynomial preprocessing techniques to achieve a query time of $N^{o(1)}$.

Below is a lemma shown along the way in [39], which we will need for our correctness proof. Essentially, it helps showing that $f(\lambda_j)$ together of the derivatives $f'(\lambda_j), \ldots, f^\ell(\lambda_j)$ are enough to uniquely determine a univariate function f with degree $k \cdot \ell - 1$ for $j \in [k]$ and $k \geq 1$.

Lemma 3.1 (Lemma 1 in [39])**.** *Let $f \in \mathbb{F}_q[X]$ where $q = p^\alpha$ for some prime p and $\alpha > 0$, and $\ell \leq p - 1$. Suppose $f(X_0) = f'(X_0) = \cdots = f^\ell(X_0) = 0$, then $(X - X_0)^{\ell+1} | f$.*

4 Our First Construction

We now move to construct our multi-server doubly efficient PIR scheme. Recall that doubly efficient refers to almost linear server preprocessing time, and sublinear client query times. Note that the scheme recalled in Section 3 does not fit the definition since the server performs $\Omega(N)$ work per query.

Then, we pose the question of whether it is possible to utilize the recent idea in [31] to enhance the scheme in [39] preprocessing the polynomial first, and the allow the servers to serve client queries in time sublinear in the database size N. In particular, in [27], the authors show how to construct a data structure that allows one to preprocess an m-variate polynomial with individual degree $< d$ such that the preprocessing takes $\Omega(d^m \cdot m^m)$ time (and storage) and the evaluation of any arbitrary input can be done in $\mathsf{poly}(m, d, \log(q))$ time (used by Lin et al. in the PIR context in [31]).

Unfortunately, naively applying this technique to the scheme by Woodruff and Yekhanin does not achieve the desired results. Specifically, in the scheme presented, the preprocessing takes roughly $O(2^{N^{1/k}})$, since $m \approx N^{1/k}$ and $d = 2$ (each variable only has degree 1) for some constant k, Therefore, the question is: can we achieve (almost) N total preprocessing time and server-side storage while achieving (almost) constant online costs (computation and communication)?

4.1 Substituting the Function

As mentioned, the major issue is that the function evaluated by the server cannot be efficiently preprocessed. Therefore, our first idea is to replace the underlying function.

As shown in [31, Claim 4.2.1], instead, we can first encode each index $i \in [N]$ into a d-ary vector $\vec{i}$ such that $i = \sum_j \vec{i}_j \cdot d^{j-1}$ (via some deterministic encoding algorithm $\mathsf{encode}(\cdot)$). Then, we can interpolate a polynomial P with m variables and individual degree d over some finite field $\mathbb{F}_q$ for $q > d$, where $d^m \geq N$, such that $P(\vec{i}) = \mathsf{DB}[i]$.

In this case, the client instead needs to interpolate a function f with degree $d \cdot m$ (since P has m variables and each variable has degree at most d). To interpolate such a function, the client needs $d \cdot m + 1$ points, which can be obtained by $d \cdot m + 1$ servers. In [31], they provide two sets of parameter settings:

1. In the first setting, $d = O(\log^{2/\epsilon} N)$ and $m = O(\epsilon \log(N)/\log\log(N))$ for some constant ϵ. In this case, the total number of servers needed is $O(\log^2(N))$ if we set $\epsilon = 2$, and the preprocessing takes $N^{1+\epsilon} = N^3$, which is $\mathsf{poly}(N)$. Alternatively, ϵ can be set smaller (e.g. 1/10), and the preprocessing becomes $N^{1.1}$ which is much closer to linear preprocessing. However, in this case, the number of servers needed is then $\log^{21}(N)$ which is quite large.

2. In the second setting, $d = 2^{O(\sqrt{\log(N)})}$ and $m = O(\sqrt{\log(N)})$. Under this setting, the preprocessing time remains $N^{1+o(1)}$ which is essentially linear. However, the total number of servers required in this case is $2^{\sqrt{\log(N)}}$, which is super-polylogarithmic, and undesirable.

Then, using this framework, it seems impossible to achieve both a polylogarithmic number of servers and an almost linear time preprocessing time. However, we notice that[11] (recalled in Theorem 2.2) reduces the preprocessing time of the scheme used in [31] from $d^m \cdot m^m$ to $d^m \cdot 4^m$. The requirement to use this scheme is that it only works with finite fields of small characteristic ($p = d^{o(1)}$).

By leveraging this new result, with a careful parameters selection we can achieve all of our desired asymptotics, (1) a polylogarithmic number of servers and (2) almost linear preprocessing time and (3) fast query times.

Specifically, we show in Theorem 4.1 that with such a new technique, and careful parameter selection, we can reduce the number of servers to $\widetilde{O}(\log(N))$, while keeping the preprocessing time $N^{1+o(1)}$, and with a query time of $N^{o(1)}$.

We formally present our scheme in Fig. 1.

Theorem 4.1. *The Private Information Retrieval scheme in Figure 1 satisfies Theorem 2.1 is doubly-efficient. In particular, it runs with efficiencies:*

- *Number of servers:* $\tilde{O}(\log(N)) = O(\frac{\log(N)\log\log(N)}{\log\log\log(N)})$.
- *Preprocess time and server storage:* $N^{1+o(1)}$.
- *Total online time:* $N^{o(1)}$.
- *Total online communication:* $N^{o(1)}$.

Proof. We prove the properties separately.

Correctness: For correctness, we simply need to argue that $f(0) = P(i)$. This is simple since we have $dm + 1$ distinct points and are recovering a function f of degree dm.

Privacy: Privacy is also simple: since $\vec{v}$ is sampled uniformly at random from $\mathbb{F}_q^m$, $j \cdot \vec{v} + \mathsf{encode}(i)$ is indistinguishable from a uniformly randomly drawn vector from $\mathbb{F}_q^m$ for any $j \in [k]$ (for $[k]$ denoting k elements of $\mathbb{F}_q$).

Efficiency: Now, to prove the efficiency, we first set the parameters accordingly.

Public Parameters (set according to the proof of Theorem 4.1):

1. m: number of variables.
2. d: total degree of a polynomial function.
3. q: a prime power p^α for some prime p and $\alpha \in \mathbb{Z}^+$.
4. $k = dm + 1$: number of servers.
5. $\mathsf{encode}(i) \to \mathbb{F}_q^m$: deterministic mapping for $i \in [N]$ to $\mathbb{Z}_q^m$.

$\underline{\mathsf{Preprocess}(\mathsf{DB})}$:

1. Interpolate m-variate, d-degree polynomial $P : \mathbb{F}_q^m \to \mathbb{F}_q$, such that $P(\mathsf{encode}(i)) = \mathsf{DB}[i]$.
2. Compute $\widetilde{P} \leftarrow \mathsf{PolyPreProcess}(P)$.
3. Return $\widetilde{\mathsf{DB}} = \widetilde{P}$.

$\underline{\mathsf{Query}(i \in [N], j \in [k] \subset \mathbb{F}_q)}$:

1. Sample $\vec{v} \leftarrow \mathbb{F}_q^m$.
2. Compute $q_j \leftarrow j \cdot \vec{v} + \mathsf{encode}(i)$. (For simplicity, we view $[k]$ as the first k elements in $\mathbb{F}_q$.)
3. Return q_j.

$\underline{\mathsf{Answer}(\widetilde{\mathsf{DB}}, q)\ :}$

1. Return $\widetilde{P}(q)$.

$\underline{\mathsf{Reconstruct}(A_1, \ldots, A_k)\ :}$

1. Use the answer to interpolate a function $f : \mathbb{F}_q \to \mathbb{F}_q$ of degree $d \cdot m$, using $f(j) = P(j \cdot \vec{v} + \mathsf{encode}(i))$ (included in A) for $j \in [k]$ where $k = dm + 1$.
2. Return $f(0)$.

Figure 1: Our multi-server doubly efficient PIR scheme.

1. We set $d^m = N$ and $d = \log\log(N) = 2^{\log\log\log(N)}$, which gives $m = O(\log(N)/\log\log\log(N))$.
2. Then, we set $p = O(\log\log\log(N)) = d^{o(1)}$ (satisfying the requirement in Theorem 2.2).
3. Lastly, we set α such that $p^\alpha = q \geq k+1$, and thus:
$$\alpha = O(\log_p(k)) = O(\log\log(N)^{1/\log\log\log\log(N)}).$$

With these parameters, we analyze each efficiency metrics separately.

- Number of servers: $k = m \cdot d = O(\log(N)\log\log(N)/\log\log\log(N)) = \tilde{O}(\log(N))$.
- Preprocessing time: Let's start with the preprocessing of the polynomial (line 2). Recall that from Theorem 2.2, the preprocessing time for a degree-d and m-variate polynomial is: $(\alpha dp)^m \cdot 4^m \cdot \mathsf{poly}(\alpha, d, m, p)$. We analyze this term by term.
  - $\alpha^m = (\log_p(k+1))^m = O(N^{1/\log\log\log\log(N)}) = N^{o(1)}$.
  - $d^m = N$.
  - $p^m = N^{\log\log\log\log(N)/\log\log\log(N)} = N^{o(1)}$.
  - $\mathsf{poly}(\alpha, d, m, p) = \mathsf{polylog}(N)$.

  Therefore, the total preprocessing time for a single function is $N^{1+o(1)}$. For $dm+1 = \mathsf{polylog}(N)$ servers, The total polynomial preprocessing time is thus $N^{1+o(1)}$.

  Now, let us move to the interpolation of the functions (line 1). As introduced in Theorem 2.3, the interpolation time is $N\mathsf{polylog}(N)$. Putting all these together, the preprocessing time is $N^{1+o(1)}$. Therefore, the extra storage (i.e., $|\widetilde{\mathsf{DB}}|$) is also trivially bounded by $N^{1+o(1)}$.
- Online time: The online time of the client query is simply the time for encoding, which is $O(\log(N))$. Then, for each server, the online time is to evaluate the preprocessed function. As in Theorem 2.2, each evaluation takes $4^m \cdot \mathsf{poly}(\alpha, d, m, p)$. Since $4^m = N^{o(1)}$ and $\mathsf{poly}(\alpha, d, m, p) = \mathsf{polylog}(N)$, the runtime of each server is $N^{o(1)}$. Since there are only $dm+1$ servers, the total runtime is also $N^{o(1)}$. Lastly, the reconstruction time is the time to reconstruct f plus the evaluation of f, which is simply bounded by $\mathsf{poly}(dm) = \mathsf{polylog}(N)$ [20]. The evaluation is also $O(dm) = \mathsf{polylog}(N)$ since the function has degree dm.

  Note that the unit of the computation cost above is the number of $\mathbb{F}_q$ operations. Note that since $q = O(\mathsf{polylog}(N))$, each $\mathbb{F}_q$ operation costs at most $\mathsf{polylog}(N)$ bit operations and thus does not affect the conclusion above.
- Online communication: The query size is simply m $\mathbb{F}_q$ elements, and thus $\mathsf{polylog}(N)$. The answer size is bounded by the total runtime of all the servers, which is thus $N^{o(1)}$. Thus, the total online communication is also $N^{o(1)}$.

With all these efficiency metrics proven, our construction indeed satisfies the definition of being *doubly efficient.* $\square$

5 Our Second Construction

Next, we explore whether we reduce the number of servers needed even further.

Recall that in [39], each server evaluates not only the interpolated function, but also the derivatives of the functions. While they only use one level of derivatives, we can naturally extend their idea to work over L levels of derivatives. In more detail, [39] shows that for an (odd) degree $2k-1$ univariate function f, it suffices to have k unique points $f(x_i) = y_i$ for $i \in [k]$ and k unique derivatives $f'(x_i) = z_i$ to uniquely determine f. We further extend their insight and show that with k points, k first derivatives, k second derivatives, and so on, up to k L-th derivatives, to uniquely determine the function with degree $k(L+1)-1$, as formally proven in the following lemma.

Lemma 5.1. *Suppose $\{x_i\}, \{y_{0,i}\}, \ldots, \{y_{L,i}\}$ are elements of some finite field $\mathbb{F}_q$, where $i \in [I]$ for some I greater than 0 and the set $\{x_i\}$ has size I; then there exists at most one polynomial $f(x) \in \mathbb{F}_q[x]$ of degree less than or equal to $(L+1) \cdot I - 1$ such that $f(x_i) = y_{0,i}$ and $f^{\ell}(x_i) = y_{\ell,i}$ for $\ell \in [L-1]$ and f^{ℓ} denotes the ℓ-th derivative of f, where $\mathbb{F}_q$ has characteristics p greater than or equal to $L+1$.*

Proof. If there exist two such polynomials $f_1(x), f_2(x)$, then let $f = f_1 - f_2$. Then, it is clear that $f(x_i) = \cdots = f^L(x_i) = 0$ for all $i \in [I]$. Thus, by Theorem 3.1, it holds that $\prod_{i\in[I]}(x-x_i)^{L+1} | f(x)$, implying $f(x) = 0$ since the degree of f is at most $|L+1|I-1$. □

With this in mind, instead, each server calculates the partial derivatives of the interpolated polynomial P, and preprocess these derivatives as a polynomial as well. More formally, each server computes $\frac{\partial^{\ell} P}{\partial x_{z_1}, \ldots x_{z_{\ell}}}$, for all $z_1, \ldots, z_{\ell} \in [m]^{\ell}$.

With L levels of derivatives, we then only need approximately dm/L servers. However, note that for each level of derivative, the number of functions that are stored at the server grows by a factor of m (as $z_1, \ldots, z_{\ell} \in [m]^{\ell}$). Thus, for L levels, there are $O(m^L)$ functions. To make the preprocessing time and online costs reasonable, we need to tune the parameters accordingly.

In Theorem 5.2, we show that with the careful parameter setup, we can achieve $o(\log(N))$ servers, while maintaining the online cost to be $N^{o(1)}$ and preprocessing time to be $\mathsf{poly}(N)$.

We formally present our construction in Fig. 2.

Theorem 5.2. *The Private Information Retrieval scheme in Figure 2 satisfies Theorem 2.1 that is query efficient. In particular, it runs with efficiencies:*

- *Number of servers: $o(\log(N))$.*
- *Preprocess time and server storage: $N^{2+o(1)}$.*
- *Total online time: $N^{o(1)}$.*
- *Total online communication: $N^{o(1)}$.*

Proof. We prove the properties separately. We start with the privacy and efficiency, since the correctness depends on the parameters setup analyzed in detail in the efficiency analysis.

Privacy: Privacy is exactly the same as in Theorem 4.1.

Efficiency: Now, to prove the efficiency, we first set the parameters accordingly.

Public Parameters (set according to the proof of Theorem 5.2):

1. m: number of variables.
2. d: total degree of a polynomial function.
3. q: a prime power p^α for some prime p and $\alpha \in \mathbb{Z}^+$.
4. k: number of servers.
5. $\mathsf{encode}(i) \to \mathbb{F}_q^m$: deterministic mapping for $i \in [N]$ to $\mathbb{Z}_q^m$.
6. L: levels of derivatives we need.

<u>$\mathsf{Preprocess}(\mathsf{DB})$:</u>

1. Interpolate m-variate, d-degree polynomial $P : \mathbb{F}_q^m \to \mathbb{F}_q$, such that $P(\mathsf{encode}(i)) = \mathsf{DB}[i]$.
2. Compute $\widetilde{P} \leftarrow \mathsf{PolyPreProcess}(P)$.
3. Compute $P_{x_{z_1},\ldots,x_{z_\ell},\ell} \leftarrow \frac{\partial^\ell P}{\partial x_{z_1},\ldots,\partial x_{z_\ell}}$ for every $\ell \in [L]$, for all $(z_1, \ldots, z_\ell) \in [m]^\ell$.
4. Compute $\widetilde{P}_{x_{z_1},\ldots,x_{z_\ell},\ell} \leftarrow \mathsf{PolyPreProcess}(P_{x_{z_1},\ldots,x_{z_\ell},\ell})$.
5. Return $\widetilde{\mathsf{DB}} = (\widetilde{P}, \widetilde{P}_{x_{z_1},\ldots,x_{z_\ell},\ell})$ for all $(z_1, \ldots, z_\ell) \in [m]^\ell$ and $\ell \in [L]$.

<u>$\mathsf{Query}(i \in [N], j \in [k] \subset \mathbb{F}_q)$:</u>

1. Sample $\vec{v} \leftarrow \mathbb{F}_q^m$.
2. Compute $q_j \leftarrow j \cdot \vec{v} + \mathsf{encode}(i)$. (For simplicity, we view $[k]$ as the first k elements in $\mathbb{F}_q$.)
3. Return q_j.

<u>$\mathsf{Answer}(\widetilde{\mathsf{DB}}, q)$:</u>

1. Retruen $\widetilde{P}(q)$ and $\widetilde{P}_{z_1,\ldots,z_\ell,\ell}(q)$ for all $(z_1, \ldots, z_\ell) \in [m]^\ell$ and $\ell \in [L]$.

<u>$\mathsf{Reconstruct}(A_1, \ldots, A_k)$:</u>

1. Use the answer to interpolate a function $f : \mathbb{F}_q \to \mathbb{F}_q$ of degree $d \cdot m$, as follows: $f(j) = P(j \cdot \vec{v} + \mathsf{encode}(i))$ (included in A_j); and

$$f^\ell(j) = \sum_{z_1,\ldots,z_\ell \in [m]^\ell} \frac{\partial^\ell P}{\partial x_{z_1}, \ldots, \partial x_{z_\ell}}(j \cdot \vec{v} + \mathsf{encode}(i)) \prod_{i \in [\ell]} \vec{v}_{z_i} \tag{1}$$

2. Return $f(0)$.

Figure 2: Our recursive PIR scheme.

1. We set $d^d = N$ which gives $d = \frac{\log(N)}{W(\log(N))}$ where $W(\cdot)$ is the Lambert W function [16].
2. Then, we set $m = d$.
3. And we set $L = \frac{\log(N)}{W(\log(N))^{1.5}}$.
4. Then, set $k = m \cdot d/L$.
5. Furthermore, we set prime p to be the smallest prime such that $p \geq L + 1$ (by Bertrand's postulate [34], $p = O(L)$).
6. Lastly, we set α such that $p^\alpha = q \geq k + 1$, which means:
$$\alpha = O(\log_p(k)) = \log_p(p \cdot W(\log(N))) = 1 + \log_p(W(\log(N))) = 1 + o(1) = O(1).$$

With these parameters, we analyze the efficiency metrics accordingly.

- Number of servers: $k = m \cdot d/L = O(\frac{\log(N)}{\sqrt{W(\log(N))}}) = o(\log(N))$.
- Preprocessing time: Let's start with the preprocessing of each polynomial (line 2 and 4). Recall that from Theorem 2.2, the preprocessing time for a single function is: $(\alpha dp)^m \cdot 4^m \cdot \mathsf{poly}(\alpha, d, m, p)$. We analyze this term by term.
  - $\alpha^m = O(1)^m = N^{o(1)}$.
  - $d^m = N$ as parametrized above.
  - $p^m = O(N)$ (since $m^m = O(N)$, $p = O(L)$ and $L = o(m)$).
  - $\mathsf{poly}(\alpha, d, m, p) = \mathsf{polylog}(N)$.

  Therefore, the total preprocessing time for a single function is $N^{2+o(1)}$. In total, there are at most:
  $$m^L + 1 = O(N^{1/\sqrt{W(\log(N))}}) = N^{o(1)}$$
  functions. Then, the total preprocessing time is $N^{2+o(1)}$.

  Now, let us move to the interpolation of the functions and the calculations of the derivatives (line 1 and 3). As introduced in Theorem 2.3, the interpolation time is $N \cdot \mathsf{polylog}(N)$. Furthermore, computing the partial derivative over a single variable of a function is trivially bounded by $O(N)$, and since there are at most $N^{o(1)}$ partial derivatives (as calculated above), the cost is $N^{1+o(1)}$. Putting all these together, the preprocessing time is $N^{1+o(1)}$. Therefore, the extra storage is also trivially bounded by $N^{2+o(1)}$.
- Online time: The online time of the client query is trivially the time for encoding, which is $\log(N)$ Then, for each server, the online time is to evaluate the preprocessed function. As in Theorem 2.2, each evaluation takes $4^m \cdot \mathsf{poly}(\alpha, d, m, p)$. Since $4^m = N^{o(1)}$ and $\mathsf{poly}(\alpha, d, m, p) = \mathsf{polylog}(N)$, and there are a total of $m^L = N^{o(1)}$ evaluations, the runtime of each server is $N^{o(1)}$. Since there are only $o(\log(N))$ servers, the total runtime is also $N^{o(1)}$. Lastly, the reconstruction time is the time to reconstruct f plus the evaluation of f. Note that there are at most $N^{o(1)}$ points, we can thus recover $f(j), f'(j), f''(j), \ldots, f^L(j)$

within time $N^{o(1)}$. Lastly, using n points, we can recover function f with time $\mathsf{poly}(n)$ [20], and $n = N^{o(1)}$. The evaluation is also $O(dm) = \mathsf{polylog}(N)$ since the function has degree dm.

Again, the unit of the computation cost above is the number of $\mathbb{F}_q$ operations. However, since $q = O(\mathsf{polylog}(N))$, each $\mathbb{F}_q$ operation costs at most $\mathsf{polylog}(N)$ bit operations and thus does not affect the calculation above.

- Online communication: The query size is simply m $\mathbb{F}_q$ elements, and thus $\mathsf{polylog}(N)$. The answer size is bounded by the total runtime of all the servers, which is $N^{o(1)}$. Therefore, the total online communication for the scheme is also $N^{o(1)}$.

Correctness: Lastly, with the parameters setup, we need to argue that $f(0) = P(i)$. It is a simple application of chain rule as described in Section 3.

As shown in Theorem 5.1 (note that we set $p \geq L + 1$ and thus $\mathbb{F}_q$ has characteristics $p \geq L + 1$), since we have k unique points of f, k corresponding first derivatives of f, and so on, up to k corresponding L-th derivatives of f, we can uniquely determine a function f with degree $d \cdot m \leq k(L+1)-1$. The derivatives are obtained by a simple application of the chain rule (Eq. (1)).

Therefore, $f(0) = P(\mathsf{encode}(i))$. □

6 Conclusion and Future Directions

In this paper, we introduce two information-theoretic PIR constructions. Our first construction is first doubly-efficient information-theoretic PIR construction: it allows the servers to preprocess a database of size N in time $N^{1+o(1)}$, and answer an infinite amount of queries in $N^{o(1)}$ time (and bandwidth), with about $O(\log(N))$ non-colluding servers. Our second construction reduces the number of needed servers to $o(\log(N))$ at the cost of increasing the preprocessing cost to $N^{2+o(1)}$.

For future work, we find the following questions to be compelling:

- (Reducing the number of servers) Can we reduce the number of non-colluding servers required to a constant number (or ideally 2 servers), while maintaining our current asymptotics (or slightly relaxing the current asymptotics, e.g., allowing the preprocessing time to be $O(N^{1+\epsilon})$ for some constant ϵ)? We know of no negative results in this regard. Nevertheless, there are no known constructions even given some that achieve even close to the asymptotics above with a small, constant number of servers.

- (Computational PIR) Can introducing (lightweight) computational assumptions (e.g. OWF) help reduce either the number of servers or the online query time for our family of schemes? In non-preprocessing PIR, distributed point functions have enabled two-server computational PIR with $O(\log N)$ communication, which has yet to be achieved for two-server information-theoretic PIR. We think it is probable that likewise, some computational assumption may help further improve the current known asymptotics for multi-server server-preprocessing PIR.

References

[1] A. Ali, T. Lepoint, S. Patel, M. Raykova, P. Schoppmann, K. Seth, and K. Yeo. {Communication–Computation} Trade-offs in {PIR}. pages 1811–1828, 2021.

[2] S. Angel and S. Setty. Unobservable communication over fully untrusted infrastructure. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, OSDI'16, pages 551–569, USA, Nov. 2016. USENIX Association.

[3] M. Backes, A. Kate, M. Maffei, and K. Pecina. ObliviAd: Provably Secure and Practical Online Behavioral Advertising. In *2012 IEEE Symposium on Security and Privacy*, pages 257–271, May 2012. ISSN: 2375-1207.

[4] A. Beimel and Y. Ishai. Information-Theoretic Private Information Retrieval: A Unified Construction. In G. Goos, J. Hartmanis, J. Van Leeuwen, F. Orejas, P. G. Spirakis, and J. Van Leeuwen, editors, *Automata, Languages and Programming*, volume 2076, pages 912–926. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[5] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond. Breaking the O(n1(2k-1)/) barrier for information-theoretic Private Information Retrieval. pages 261–270, Feb. 2002.

[6] A. Beimel, Y. Ishai, and T. Malkin. Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing. In M. Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, Lecture Notes in Computer Science, pages 55–73, Berlin, Heidelberg, 2000. Springer.

[7] A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: Pir with preprocessing. *Journal of Cryptology*, 17:125–151, 2004.

[8] A. Beimel and Y. Stahl. Robust Information-Theoretic Private Information Retrieval. *Journal of Cryptology*, 20(3):295–321, July 2007.

[9] S. Ben-David, Y. T. Kalai, and O. Paneth. Verifiable Private Information Retrieval. In *IACR TCC 2022*, 2022. Report Number: 1560.

[10] V. Bhargava, S. Ghosh, Z. Guo, M. Kumar, and C. Umans. Fast Multivariate Multipoint Evaluation Over All Finite Fields. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 221–232, Oct. 2022. ISSN: 2575-8454.

[11] V. Bhargava, S. Ghosh, M. Kumar, and C. K. 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, pages 403–415, New York, NY, USA, June 2022. Association for Computing Machinery.

[12] E. Boyle, N. Gilboa, and Y. Ishai. Function Secret Sharing. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, Lecture Notes in Computer Science, pages 337–367, Berlin, Heidelberg, 2015. Springer.

[13] E. Boyle, Y. Ishai, R. Pass, and M. Wootters. Can We Access a Database Both Locally and Privately? pages 662–693, Nov. 2017.

[14] B. Chor, N. Gilboa, and M. Naor. Private Information Retrieval by Keywords, 1998. Report Number: 003.

[15] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. pages 41–50, 1995.

[16] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth. On the lambert w function. *Advances in Computational Mathematics*, 5:329–359, 01 1996.

[17] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan. Single-Server Private Information Retrieval with Sublinear Amortized Time. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part II*, pages 3–33, Berlin, Heidelberg, May 2022. Springer-Verlag.

[18] H. Corrigan-Gibbs and D. Kogan. Private Information Retrieval with Sublinear Online Time. In A. Canteaut and Y. Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, Lecture Notes in Computer Science, pages 44–75, Cham, 2020. Springer International Publishing.

[19] B. Fisch, A. Lazzaretti, Z. Liu, and C. Papamanthou. Single server pir via homomorphic thorp shuffles. Cryptology ePrint Archive, Paper 2024/482, 2024. `https://eprint.iacr.org/2024/482`.

[20] J. V. Z. Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, USA, 2 edition, 2003.

[21] A. Ghoshal, B. Li, Y. Ma, C. Dai, and E. Shi. Information-theoretic multi-server pir with global preprocessing. Cryptology ePrint Archive, Paper 2024/765, 2024. `https://eprint.iacr.org/2024/765`.

[22] N. Gilboa and Y. Ishai. Distributed Point Functions and Their Applications. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, Lecture Notes in Computer Science, pages 640–658, Berlin, Heidelberg, 2014. Springer.

[23] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish. Scalable and private media consumption with Popcorn. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 91–107, USA, Mar. 2016. USENIX Association.

[24] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. page 27, 2022.

[25] L. Hetz, T. Schneider, and C. Weinert. Scaling Mobile Private Contact Discovery to Billions of Users, 2023. Publication info: Published elsewhere. Minor revision. ESORICS 2023.

[26] J. Holmgren, R. Canetti, and S. Richelson. Towards Doubly Efficient Private Information Retrieval. Technical Report 568, 2017.

[27] K. S. Kedlaya and C. Umans. Fast Polynomial Factorization and Modular Composition. *SIAM Journal on Computing*, 40(6):1767–1802, Jan. 2011. Publisher: Society for Industrial and Applied Mathematics.

[28] D. Kogan and H. Corrigan-Gibbs. Private Blocklist Lookups with Checklist. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 875–892. USENIX Association, 2021.

[29] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373, Miami Beach, FL, USA, 1997. IEEE Comput. Soc.

[30] A. Lazzaretti and C. Papamanthou. Near-Optimal Private Information Retrieval with Preprocessing. In G. Rothblum and H. Wee, editors, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 406–435, Cham, 2023. Springer Nature Switzerland.

[31] W.-K. Lin, E. Mook, and D. Wichs. Doubly Efficient Private Information Retrieval and Fully Homomorphic RAM Computation from Ring LWE, 2022. Report Number: 1703.

[32] S. J. Menon and D. J. Wu. Spiral: Fast, High-Rate Single-Server PIR via FHE Composition. In *IEEE Symposium on Security and Privacy, 2022*, 2022.

[33] M. H. Mughees, H. Chen, and L. Ren. OnionPIR: Response Efficient Single-Server PIR. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 2292–2306, New York, NY, USA, Nov. 2021. Association for Computing Machinery.

[34] T. Nagell. *Introduction to Number Theory.* American Mathematical Soc., 1964. Google-Books-ID: Znc5EAAAQBAJ.

[35] H. Okada, R. Player, S. Pohmann, and C. Weinert. Towards Practical Doubly-Efficient Private Information Retrieval, 2023. Publication info: Published elsewhere. Minor revision. Financial Cryptography and Data Security 2024.

[36] G. Persiano and K. Yeo. Limits of Preprocessing for Single-Server PIR. Technical Report 235, 2022.

[37] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, Nov. 1979.

[38] E. Shi, W. Aqeel, B. Chandrasekaran, and B. Maggs. Puncturable Pseudorandom Sets and Private Information Retrieval with Near-Optimal Online Bandwidth and Time. In *Advances in Cryptology - CRYPTO*, 2021.

[39] D. Woodruff and S. Yekhanin. A Geometric Approach to Information-Theoretic Private Information Retrieval. 2005.

[40] S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. *Journal of the ACM*, 55(1):1–16, Feb. 2008.

[41] S. Yekhanin. *Locally Decodable Codes and Private Information Retrieval Schemes.* Information Security and Cryptography. Springer, Berlin, Heidelberg, 2010.

[42] L. Zhao, X. Wang, and X. Huang. Verifiable single-server private information retrieval from LWE with binary errors. *Information Sciences*, 546:897–923, Feb. 2021.

[43] M. Zhou, W.-K. Lin, Y. Tselekounis, and E. Shi. Optimal Single-Server Private Information Retrieval. *ePrint IACR*, 2022.