Information-Theoretic Single-Server PIR in the Shuffle Model

Yuval Ishai
Technion, Haifa, Israel

Mahimna Kelkar
Cornell University, New York, NY, USA

Daniel Lee
MIT, Cambridge, MA, USA

Yiping Ma
University of Pennsylvania, Philadelphia, PA, USA

Abstract

We revisit the problem of private information retrieval (PIR) in the *shuffle model*, where queries can be made anonymously by multiple clients. We present the first single-server PIR protocol in this model that has sublinear per-client communication and *information-theoretic security*. Moreover, following one-time preprocessing on the server side, our protocol only requires sublinear per-client *computation*. Concretely, for every $\gamma > 0$, the protocol has $O(n^\gamma)$ communication and computation costs per (stateless) client, with $1/\mathsf{poly}(n)$ statistical security, assuming that a size-n database is simultaneously accessed by $\mathsf{poly}(n)$ clients. This should be contrasted with the recent breakthrough result of Lin, Mook, and Wichs (STOC 2023) on doubly efficient PIR in the standard model, which is (inherently) limited to computational security.

2012 ACM Subject Classification Security and privacy → Information-theoretic techniques

Keywords and phrases Private information retrieval, Shuffle model

Digital Object Identifier 10.4230/LIPIcs.ITC.2024.6

Related Version *Full Version*: https://eprint.iacr.org/2024/930

Funding This research was supported by a Google faculty grant. Yuval Ishai was additionally supported by ERC Project NTSC (742754), BSF grants 2018393 and 2022370, ISF grant 2774/20, and ISF-NSFC grant 3127/23. Mahimna Kelkar and Yiping Ma were partially supported by a Technion research scholarship. Yiping Ma was also supported by a Microsoft Research PhD Fellowship.

1 Introduction

A private information retrieval (PIR) protocol [15, 36] allows a client to fetch an entry from a database server without revealing which entry was fetched. Specifically, the server holds a database $x = (x_1, \ldots, x_n)$ consisting of n bits (or generically, n symbols over an alphabet Σ) while the client holds an index $i \in \{1, \ldots, n\}$; the client wishes to obtain x_i while hiding i from the server.

PIR protocols have been broadly studied in two flavors: information-theoretic and computational. Information-theoretic protocols provide security against computationally unbounded adversaries and do not require "cryptographic" computations. Unfortunately, non-trivial information-theoretic PIR (with less than n bits of communication) is impossible given only one server [15]. Consequently, PIR protocols in this setting need database replication across two or more non-colluding servers. This poses challenges for deployment since the cost of managing multiple storage spots is high when databases are large (e.g., synchronization, monetary cost), and enforcing non-collusion on the database servers is

hard in practice, especially when the data is owned by a single entity (e.g., a company). In contrast, computational PIR can work when only one server holds the database but only provides security against polynomial-time adversaries due to its reliance on cryptographic hardness assumptions (e.g., quadratic residuosity, learning with errors). Furthermore, the associated cost is typically high due to expensive cryptographic operations at the server – indeed, existing single-server protocols [3, 5, 6, 17, 29, 38] are significantly less efficient in practice than the multi-server information-theoretic ones [26, 27].

The shuffle model: PIR with many clients. Achieving the best of both worlds, as aforementioned, is not possible in the standard model without using n bits of communication. To circumvent this barrier, Ishai, Kushilevitz, Ostrovsky and Sahai [32] proposed a relaxed model, where *many* clients (with arbitrarily correlated indices) simultaneously query a single server, but the clients are granted the ability to make anonymous queries to the server. Abstractly, we can think of the queries as being *shuffled* before reaching the server.

Specifically, consider a client using a *multi-server* PIR query algorithm to generate sub-queries for a query index. If these sub-queries were naively sent to a *single server*, the server would immediately learn the query index of this client. However, this work and [32] show the power of shuffling: if there are many clients and their sub-queries are randomly permuted by a shuffler before being sent to the server, then it is hard for the server – even one that is computationally unbounded as we show in this work – to figure out any of the client-query indices. Therefore, this single server in the shuffle model can simply perform "cheap" operations of the multi-server PIR scheme to answer sub-queries.

Understanding *the shuffle model* in the context of PIR is well-motivated by real-world applications: databases with high-volume queries, such as stock quotes and search engines, naturally enjoy the feature that thousands of users access the databases at the same time, and therefore considering PIR with many simultaneously querying clients is sensible, particularly if it allows for cheaper server cost. Note that this is a substantially different goal from batch PIR [28, 31] which amortizes the cost of multiple queries from a *single client*.

The shuffle model has been considered also in problems orthogonal to PIR, including secure aggregation [8, 23, 32] and differential privacy [4, 10, 13, 14, 22]. Analogously to these works, we view shuffling as an *atomic* operation; existing literature on differential privacy [8] and anonymity [1, 18, 30, 37, 41] discusses how to implement shuffling efficiently (see details in Section 1.2).

The shuffle PIR model opens a promising direction toward constructing efficient single-server PIR protocols. In this work, we establish the *theoretical feasibility* of non-trivial single-server PIR with information-theoretic security in the shuffle model.

1.1 Our Results

This paper aims to develop a formal understanding of PIR in the shuffle model from a theoretical perspective. We briefly detail our results below.

Information-theoretic single-server PIR in the shuffle model (Sections 5 and 6). We present the first construction for single-server PIR in the shuffle model that has *sublinear communication* and *information-theoretic* security (with inverse-polynomial statistical error). Moreover, our construction is also *doubly efficient*: following one-time preprocessing on the server side, and without any state information on the client side, the server's per-query computation is sublinear in the database size.

▶ **Theorem 1** (Informal). *For every constant $0 < \gamma < 1$, there exists a single-server PIR protocol in the shuffle model such that, on database of size n, and following one-time preprocessing on the server side, the protocol has $O(n^{\gamma})$ per-query computation and communication, and $O(n^{1+\gamma/2})$ server storage. This is achieved with the following information-theoretic security guarantee: for any inverse polynomial $\epsilon = 1/p_1(n)$, there exists a polynomial $p_2(n) = O(n^{1+4/\gamma} \cdot (p_1(n))^8)$ such that the protocol has ϵ-statistical security as long as the total number of queries made by (uncorrupted) clients is at least $p_2(n)$.*

As a key technique, we describe a generic *inner-outer* paradigm that composes together two standard (multi-server) PIR protocols: an outer and an inner layer, to build a PIR protocol in the shuffle model. Besides, our results are robust against imperfect shuffling/anonymity (details in the full version).

While the above protocol only achieves inverse-polynomial (rather than negligible) security error, this is in fact the standard notion of security in several important settings, including differential privacy [19,21], secure computation with partial fairness [16,24,39], and secure computation over one-way noisy communication [2]. Our protocol demonstrates that *information-theoretic* security is indeed feasible *without database replication*. Moreover, while concrete efficiency is not the focus of this work, we believe that our approach has potential for reducing the cost of standard-model PIR when properly combined with single-server schemes and settling for a constant-factor cost reduction that might be significant in practice (see Section 6.4 for details).

Lower bound on security (Section 6.5). In the inner-outer paradigm, we show a security lower bound when *any* generic PIR protocol is used as the outer layer, and a *constant*-server PIR from a broad class is used as the inner layer; in particular, $1/\mathsf{poly}(n)$ statistical security is tight in the sense that negligible security error cannot be achieved with polynomially many clients. We also discuss open problems (Section 7) on whether negligible security is possible (with polynomially many clients) by using other protocols in the inner layer.

1.2 Discussion on the Shuffle Model

Two-way shuffling. In the problems such as secure aggregation and differential privacy with shuffle model, the shuffled messages are delivered to a server for analytics. Our PIR setting is a bit different, since responses need to be communicated back from the server to the client, we require the shuffling to be *two-way*. Specifically, we require not only that clients can send messages anonymously to the server but also that the server can respond to clients while still keeping the client identities hidden. It is important to note that shuffling or anonymity does not trivialize the problem; it hides who sends the messages but not the content of the messages. In practice, this two-way shuffling can be realized in a number of ways [7,8,12,18,35], even without computational assumptions.

A hybrid model. PIR in the shuffle model can also be equivalently viewed as a hybrid model between the standard single-server and multi-server PIR models: as an abstraction, the shuffler models a second "server" which is assumed to not collude with the main database server but does not hold a copy of the database and can only perform *database-irrelevant* computations. This alone makes the shuffle model interesting for practical deployments: non-collusion between two (or more) servers holding the same database can be difficult to enforce (since it is likely for them to be operated by the same company for data ownership reasons) making it a strong assumption in practice; in contrast, if only one server holds

the database, then the "two" servers can be reasonably run by independent (and possibly geographically distributed) entities. We also note that it could be interesting to let this second database-irrelevant server perform more generic computations instead of just acting as a shuffler; we leave this exploration to future work.

2 Technical Overview

In this section, we present a toy protocol, which is insecure but conveys our core ideas; we then outline the techniques for building our eventual protocol from the toy protocol.

An insecure toy protocol. The starting point is the classic two-server information-theoretic PIR scheme by Beimel et al. [9]. In this scheme, a client first deterministically encodes its queried index $i \in [n]$ to a bit string $\mathbf{z}$ of length $m = O(\log n)$ (we call $\mathbf{z}$ the encoding of queried index, or simply query), and splits $\mathbf{z}$ to two *additive* shares in $\mathbb{F}_2^m$, $\mathbf{z}_1$ and $\mathbf{z}_2$ (we call them sub-queries), and then sends them to the two servers respectively.

We construct PIR in the shuffle model based on this protocol. Abstractly, each client generates two sub-queries (or shares) $\mathbf{z}_1$ and $\mathbf{z}_2$ as if it was querying using the above two-server scheme but in fact sends both sub-queries to a single server through an anonymous channel (which shuffles the sub-queries together with that from many other clients). Observe that this is exactly an instance of secure aggregation in the split-and-mix approach [8, 23, 32], where each input is split into two shares; the hope is that the server would learn nothing given the shuffled encoding shares from many clients.

There are two issues with this toy protocol. The first issue is obvious – the server learns the sum of all the encoding strings, and therefore can easily distinguish two sets of query indices by comparing the sum of their shares and the sum of their encodings. Note that leaking the sum to the server is exactly the goal of secure aggregation, but the sum should not be leaked in the PIR context. This leakage can be easily eliminated by letting one of the clients add a dummy share (a random string) to hide the sum. The second issue is more involved. In fact, splitting each input into only two shares is not enough to guarantee security; this can be demonstrated through a simple counter-example: suppose that the server wishes to distinguish between the 2-additive shares of zeros and that of ones (sharing over $\mathbb{F}_2$) . In the latter case, there is always an equal number of ones and zeros in the shares, while this is not true for the former case. This approach can be generalized to a "counting" based strategy (for sharing over any Abelian groups) and allows for generic efficient distinguishing attacks (details in the full version). While splitting into more additive shares, e.g., 4, is sufficient [8], this means we need a 4-server PIR (that has additive sub-queries) and thus leads to worse communication – $O(n^{3/4})$ in the 4-server scheme compared to $O(n^{1/2})$ in the two-server scheme (Section 4.1). On the road map to our general protocol with $O(n^\gamma)$ communication (for any $\gamma > 0$), the first checkpoint is to bypass the above attack and achieve a protocol with $O(n^{1/2})$ communication; it turns out that the key ideas used for this also play a pivotal role in our final protocol design.

Randomizing inputs via the inner-outer paradigm. The core reason why the simple split-and-mix approach does not work with two additive shares is the presence of arbitrary correlation among the queries; indeed, if all queries were independent and uniformly random, then using two shares works perfectly. Our key insight to navigate around this is to randomize the queries using another PIR, resulting in *uniform random but pairwise independent* queries which is later shown to be sufficient for security.

Our construction employs a novel approach – the *inner-outer* paradigm, which composes a k-server PIR protocol as an *outer* layer with the previous 2-server PIR protocol (with 2-additive shares) as the inner layer. At a high level, the outer layer PIR randomizes the client queries before they get processed through the inner layer PIR. Below we call the outer layer protocol as OPIR and the inner layer protocol as IPIR.

Formally, the composition works as follows: for any database $x \in \{0,1\}^n$, on input an arbitrary query index $i \in [n]$, the client first runs the OPIR query algorithm to generate k queries $q_1, \ldots, q_k$; note that they naturally satisfy pairwise independence and each is uniformly random in the OPIR query space $\mathcal{Q}$, simply because of the security property of any PIR. Instead of sending them directly to the server, these queries are *interpreted as indices* to a new database x' of size $|\mathcal{Q}|$, where x' consists of the answers to all the possible OPIR queries (i.e., elements in $\mathcal{Q}$). Now the client runs IPIR query algorithm on the each of the k "indices" in $\{1, 2, \ldots, |\mathcal{Q}|\}$, and sends the IPIR sub-queries to the server. Specifically, the client maps an index to its encoding in the two-server protocol, and splits the encoding into 2 additive shares (sub-queries) in $\mathbb{F}_2^m$ where $m = O(\log |\mathcal{Q}|)$. Finally, to have the compilation work, the server needs to build the database x' for IPIR in advance, which is feasible as long as $|\mathcal{Q}|$ is polynomial in n.

The upshot of this compilation is that the server now sees a set of shuffled shares generated from uniformly random and pairwise independent query indices to the database x'. As we shall show next, this randomization achieves that, for any two multi-sets of queried indices I, I' with distance at most δ, the resulting multi-sets after processing through OPIR will be J, J' will have distance in expectation $\sqrt{\delta}$, even though J, J' are larger than I, I'. The distance further decreases to $\sqrt[4]{\delta}$ after processing through IPIR (additive sharing). We will show that having each client add only one random noise sub-query (on top of its real sub-queries) is sufficient to hide the $\sqrt[4]{\delta}$ distance from the server.

Analyzing split and mix with pairwise independence. We now analyze the split and mix approach for pairwise independence queries which we get from the OPIR; we use a balls-and-bins formulation for this analysis. Specifically, the OPIR queries of all clients can be viewed as throwing $B = k \cdot C$ balls randomly into $|\mathcal{Q}|$ bins where C is the number of clients and $\mathcal{Q}$ is the OPIR query space. Since the balls are pairwise independent, we can bound the expected difference in the balls-and-bins configuration from any two such distributions by $\Theta_{|Q|}(\sqrt{B})$. This implies that the *differences* between any two sets of B OPIR sub-queries (and consequently, the query indices in IPIR) is proportional to $\sqrt{B}$.

As a second step, we show once again using a balls-and-bins formulation that for any sets of IPIR indices with difference δ, when the indices are split into two shares, the expected difference is proportional to $\sqrt{\delta}$. This implies that any two sets of original client indices, once put through both OPIR and IPIR, will differ on expectation by $\sqrt[4]{B}$. Our final step shows that adding just 1 noise query per client results in being able to "hide" this $\sqrt[4]{B}$ difference in order to get $1/\mathsf{poly}(n)$ security. This analysis goes through as long as the total number of clients C is at least $\Omega(n^{5+c})$ for some constant $c > 0$. More details are provided in Section 6.1. We also show a concrete instantiation using a Reed-Solomon code based OPIR.

Improving communication using CNF-shares. Following this, in Section 6.2, we show how a CNF-sharing based construction can be used as the IPIR to reduce the communication complexity; in particular, using an s-CNF sharing allows us to reduce the communication cost to $O(n^{1/s})$ given $\Omega(n^{2s+1}/\epsilon^8)$ clients for statistical security ϵ. This cleanly generalizes our earlier construction. The security proof follows a similar outline as before but is somewhat

more involved. We find a nice group theoretic formulation of the problem of understanding the symmetries within the CNF-sharing, which allows us to greatly simplify the analysis by leveraging simple results from that domain.

Lower bound on security. We show a lower bound on security for protocols within our inner-outer paradigm, by showing that negligible statistical distance cannot be achieved in this realm. To prove this, we borrow an idea from Ghazi et.al [23, Theorem 6], and extend their results on secret sharing to the PIR context. We observe that the query algorithms of multi-server PIR protocols can be viewed as secret sharing; this allows us to show that if the total number of possible ways to secret share a query index is $K = p_1(n)$ and there are $C = p_2(n)$ clients, then there must exist two sets of input indices with some $1/p_3(n)$ statistical distance, where p_1, p_2, p_3 are all polynomials in n.

3 Related Work

We note that the shuffle PIR model substantially differ from standard PIR models in the literature; the only other relevant work in this model is by Ishai et al. [32]. In this section, we discuss models and techniques specifically related to shuffling, and defer a longer comprehensive literature review on PIR to our full version.

Differential privacy (DP) for PIR. A line of work [4, 40] considers the DP notion for PIR assuming client anonymity. Here, clients send their query indices via onion routing to the server, and privacy is guaranteed by the shuffling of client indices along with some noise queries. Here DP guarantees that the server cannot distinguish neighboring sets of queries (i.e., differing in exactly one client). Unfortunately, DP is substantially *weaker* than standard PIR security and therefore insufficient in any application where client queries can be *arbitrarily correlated*, as evidenced by several works which show how sensitive information can be extracted through *frequency analysis*-based attacks [25, 34, 42].

The "Split and mix" technique. A core idea in our construction follows from an ingenious split-and-mix approach for secure summation by Ishai et al. [32]. Specifically, they give a one-round single-server secure aggregation protocol as follows: Each client *splits* its input into k additive shares; then, as part of the shuffle model, these shares from all the C clients are *mixed* together before being sent to the aggregation server who simply outputs the sum of all the shares. The security goal here is that server cannot infer anything about a particular client's input. More precisely, the shuffled shares of any two tuples of client inputs (with equal sum) should look indistinguishable. Ishai et al. [32] show that statistical security of $2^{-\sigma}$ can be achieved by using per-input $k = \Theta(\log C + \log p + \sigma)$ additive shares over a group of size p. Recent works [8, 23] improve this bound to $k = \lceil 2 + \frac{2\sigma + \log_2(p)}{\log_2(C)} \rceil$ and show that at least 4 shares are necessary.

In our shuffle PIR context, we find that 2 additive shares are sufficient due to our query randomization technique and the usage of additional *noise* queries; this cannot be done in the summation setting as the final output could change. Towards reducing the communication of our PIR protocol, we also generalize the split-and-mix approach to CNF shares.

4 Preliminaries

Basic notation. For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. $\mathbb{F}$ denotes a finite field. $\mathfrak{S}_c$ denotes the symmetric group containing all permutations of c elements. We use bold letters to denote vectors (e.g., $\mathbf{z}$). We use $\mathsf{SD}(\mathcal{D}_1, \mathcal{D}_2)$ to denote the statistical distance between the distributions $\mathcal{D}_1$ and $\mathcal{D}_2$.

Unless specified, logarithms are taken to the base 2. The notation $\mathsf{poly}(\cdot)$ refers to a fixed but unspecified polynomial in its parameter; we use $\mathsf{polylog}(\cdot)$ to mean $\mathsf{poly}(\log(\cdot))$. The notation $\widetilde{O}$ hides arbitrary polylogarithmic factors.

We use $\xrightarrow{\$}$ to denote uniformly random sampling, $\to$ for output by deterministic algorithms, and $\xrightarrow{\$}$ for output by randomized algorithms.

4.1 Multi-Server Information-Theoretic PIR

We begin with the standard notion of multi-server information-theoretic PIR below.

▶ **Definition 2** (PIR). *Let Σ be a finite alphabet. A k-server PIR protocol over Σ is a tuple $\Phi = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Recon})$ with the following syntax:*

- $\mathsf{Setup}(x) \to P_x$*: a deterministic algorithm executed by all servers that takes in an n-entry database $x \in \Sigma^n$ and outputs its encoding P_x.*
- $\mathsf{Query}(i; n) \xrightarrow{\$} ((q_1, \ldots, q_k), \mathsf{st})$*: a randomized algorithm (parameterized by n) executed by the client that takes in an index $i \in [n]$, and outputs sub-queries $q_1, \ldots, q_k$ and a state st. The sub-query q_ℓ is sent to the ℓ-th server.*
- $\mathsf{Answer}^\ell(P_x, q_\ell) \to a_\ell$*: a deterministic algorithm executed by the ℓ-th server that takes in the encoding P_x and a sub-query q_ℓ, and outputs an answer a_ℓ. Since the Answer algorithm may be different for different servers, we use ℓ to denote the algorithm used by server ℓ.*
- $\mathsf{Recon}((a_1, \ldots, a_k), \mathsf{st}) \to x_i$*: a deterministic algorithm executed by the client that takes in answers $a_1, \ldots, a_k$ (where a_ℓ is from the ℓ-th server) and the state st, and outputs $x_i \in \Sigma$.*

Φ needs to satisfy the following correctness and security properties:
Correctness. For all $n \in \mathbb{N}$, any database $x = (x_1, \ldots, x_n) \in \Sigma^n$, and all $i \in [n]$,

$$\Pr\left[\mathsf{Recon}((a_1, \ldots, a_k), \mathsf{st}) = x_i : \begin{array}{rcl} P_x & \leftarrow & \mathsf{Setup}(x) \\ ((q_1, \ldots, q_k), \mathsf{st}) & \xleftarrow{\$} & \mathsf{Query}(i; n) \\ (a_1, \ldots, a_k) & \leftarrow & (\mathsf{Answer}^\ell(P_x, q_\ell))_{\ell=1}^k \end{array}\right] = 1.$$

Intuitively, correctness says that the client always gets the correct value of x_i.
Security. For all $n \in \mathbb{N}$, $i \in [n]$, and $T \subset [k]$, define the distribution

$$\mathcal{D}_n(i, T) := \{\{q_\ell\}_{\ell \in T} : ((q_1, \ldots, q_k), \mathsf{st}) \xleftarrow{\$} \mathsf{Query}(i; n)\}.$$

We say that Φ has (t, ϵ)-privacy (where $t < k$, and $\epsilon = \epsilon(n)$), if for all $n \in \mathbb{N}$, any two indices $i, i' \in [n]$, and any set $T \subset [k]$ such that $|T| < t$, we have

$$\mathsf{SD}(\mathcal{D}_n(i, T), \mathcal{D}_n(i', T)) \le \epsilon(n).$$

Intuitively, (t, ϵ)-privacy says that any set of less than t colluding servers has a distinguishing advantage at most ϵ.

We provide as background (Appendix A), common PIR schemes that will be important for our construction for PIR in the shuffle model. The constructions employ the following general outline: The servers encode the database $x \in \Sigma^n$ as a polynomial P_x. To query the database at position i, the client first encodes i into a vector $\mathbf{z}^{(i)}$ where the encoding is defined in a way that results in $P_x(\mathbf{z}^{(i)}) = x_i$. The client now evaluates P_x at $\mathbf{z}^{(i)}$ while hiding $\mathbf{z}^{(i)}$ from the servers: it secret shares $\mathbf{z}^{(i)}$ into k shares, and each share is sent to one of the k servers (through e.g., additive or Shamir sharing). Each server can then evaluate P_x on one share and send the result to the client, who is able to reconstruct the entry x_i.

Other notation. For a PIR protocol Φ, we use $\mathcal{E}_\Phi$ to denote the encoding space of all indices. We use $\mathcal{Q}_\Phi$ to denote the space of all possible sub-queries (note that $\mathcal{Q}_\Phi$ may not equal $\mathcal{E}_\Phi$). For example, in the two-server construction above, $\mathcal{E}_\Phi$ contains all binary strings with Hamming weight d, and the space $\mathcal{Q}_\Phi$ is $\mathbb{F}_2^m$, i.e, in this case $\mathcal{E}_\Phi \subset \mathcal{Q}_\Phi$.

4.2 Balls and Bins

We formulate the core analysis of our constructions using the widely-used balls-and-bins problem, which we provide background and notation for here. Abstractly, the balls-and-bins problem analyzes the distribution of B (identical) balls thrown into N bins according to some distribution D (often independent and uniformly at random). To denote a final configuration of balls, we use a N-length vector $\mathbf{u} = (u_0, \ldots, u_{N-1})$ where u_i denotes the number of balls in bin i. We say that $\mathbf{u} = (u_0, \ldots, u_{N-1})$ is (B, N)-valid if each $u_i \in \mathbb{Z}^{\geq 0}$ and $\sum_i u_i = B$. Since our analysis often deals with sharing over a group $\mathbb{G}$, we may also label the bins using elements from $\mathbb{G}$; when $\mathbb{G}$ is unspecified, it is taken to be $\mathbb{Z}_N$.

▶ **Definition 3.** *Given (B, N)-valid configurations $\mathbf{u} = (u_0, \ldots, u_{N-1})$ and $\mathbf{v} = (v_0, \ldots, v_{N-1})$, we define the following useful terms:*

- *The edit distance, denoted by $\mathsf{ED}(\mathbf{u}, \mathbf{v})$ is defined as $\mathsf{ED}(\mathbf{u}, \mathbf{v}) = \frac{1}{2}\sum_{i=0}^{N-1} |u_i - v_i|$. Intuitively, this denotes the number of balls that need to be moved to convert $\mathbf{u}$ to $\mathbf{v}$. Note that the distance is symmetric since $\mathsf{ED}(\mathbf{u}, \mathbf{v}) = \mathsf{ED}(\mathbf{v}, \mathbf{u})$. The edit distance between two distributions $\mathcal{U}$ and $\mathcal{V}$, denoted by $\mathsf{ED}(\mathcal{U}, \mathcal{V})$; can now be defined as $\mathbb{E}_{\mathbf{u}\sim\mathcal{U},\mathbf{v}\sim\mathcal{V}}[\mathsf{ED}(\mathbf{u}, \mathbf{v})]$.*
- *The ball-intersection $\mathbf{u} \sqcap \mathbf{v}$ is $(c_0, \ldots, c_{N-1})$ where each $c_i = \min(u_i, v_i)$.*
- *The ball-difference $\mathbf{u} \ominus \mathbf{v}$ is $(u'_0, \ldots, u'_{N-1})$ where each $u'_i = \max(0, u_i - v_i)$.*

5 Single-Server PIR in the Shuffle Model: Definitions and Preliminary Results

We now formally define single-server PIR in the shuffle model, which considers many query-making clients. Importantly, no coordination is assumed among clients.

▶ **Definition 4** (PIR in the shuffle model)**.** *Let Σ be a finite alphabet. A (single-server) PIR protocol (over Σ) in the shuffle model is a tuple $\mathsf{ShPIR} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Recon})$ with a syntax similar to that of a k-server PIR (Definition 2) except for a few changes given below:*

- *$\mathsf{Setup}(x) \to P_x$: a deterministic algorithm executed by the server that takes in an n-entry database $x \in \Sigma^n$ and outputs its encoding P_x.*
- *$\mathsf{Query}(i; n) \xrightarrow{\$} (q_1, \ldots, q_k)$: a randomized algorithm (parameterized by n) executed by the client that takes in an index $i \in [n]$, and outputs sub-queries $q_1, \ldots, q_k$. Unlike in Definition 2, k may be a function of n; this is possible since the shuffle model does not require k physical servers. Further, all sub-queries will be sent to the same server. For simplicity, here we omit the state in Definition 2.*

- $\mathsf{Answer}(P_x, q_\ell) \to a_\ell$*: a deterministic algorithm executed by the server that takes in the encoding* P_x *and a sub-query* q_ℓ*, and outputs an answer* a_ℓ*. Unlike in Definition 2, there is a single* Answer *algorithm.*
- $\mathsf{Recon}(a_1, \ldots, a_k) \to x_i$*: a deterministic algorithm executed by the client that takes in answers* $a_1, \ldots, a_k$*, where for all* $\ell \in [k]$*,* a_ℓ *is the answer to the client's sub-query* q_ℓ*; and outputs* $x_i \in \Sigma$*.*

ShPIR *needs to satisfy the following correctness property:*

Correctness. *For all* $n \in \mathbb{N}$*, database* $x = (x_1, \ldots, x_n) \in \Sigma^n$*, and* $i \in [n]$*,*

$$\Pr\left[\mathsf{Recon}(a_1, \ldots, a_k) = x_i : \begin{array}{rcl} P_x & \leftarrow & \mathsf{Setup}(x) \\ (q_1, \ldots, q_k) & \leftarrow\!\!\$ & \mathsf{Query}(i; n) \\ (a_1, \ldots, a_k) & \leftarrow & (\mathsf{Answer}(P_x, q_\ell))_{\ell=1}^k \end{array}\right] = 1.$$

ShPIR *also needs to satisfy the following security property in the model where client queries are shuffled before being sent to the server.*

Security. *We will parameterize security by a shuffler* Π *and a minimum number of honest client queries* C*. Formally, let* $\Pi = \{\Pi_c\}_{c \in \mathbb{N}}$ *be an ensemble such that* Π_c *is a distribution over the symmetric group* $\mathfrak{S}_c$*. When* Π *is unspecified, we assume that each* Π_c *is a uniform distribution over* $\mathfrak{S}_c$*; we refer to this as the uniform or perfect shuffler. We discuss imperfect shufflers in the full version.*

For a given n*,* Π*, and* C*, and given a tuple* $I = (i_1, \ldots, i_C) \in [n]^C$ *of client query indices, define the distribution*

$$\widetilde{\mathcal{D}}_{n,\Pi,C}(I) = \left\{ \pi(\mathbf{q}) : \begin{array}{l} (q_1^{(1)}, \ldots, q_k^{(1)}) \leftarrow\!\!\$\ \mathsf{Query}(i_1; n) \\ \cdots \\ (q_1^{(C)}, \ldots, q_k^{(C)}) \leftarrow\!\!\$\ \mathsf{Query}(i_C; n) \\ \mathbf{q} \leftarrow (q_1^{(1)}, \ldots, q_k^{(1)}, \ldots, q_1^{(C)}, \ldots, q_k^{(C)}) \\ \pi \xleftarrow{\$} \Pi_{kC} \end{array} \right\}.$$

Then, we say that ShPIR *is* (Π, C, ϵ)*-secure if for every* $n \in \mathbb{N}$ *and all* $C^* \geq C(n)$*, and* $I, I' \in [n]^{C^*}$*, it holds that:*

$$\mathsf{SD}(\widetilde{\mathcal{D}}_{n,\Pi,C^*}(I), \widetilde{\mathcal{D}}_{n,\Pi,C^*}(I')) \leq \epsilon(n).$$

Efficiency metrics. We measure the efficiency of PIR constructions in the shuffle model using a few metrics below. Since we consider many clients querying the server, we will characterize the cost per query.

- *Per-query server computation*: for answering each query, the number of bits that the server reads from the database and the preprocessing bits.
- *Per-query communication*: the sizes of the client query and the server response.
- *Server storage*: the total number of bits, including the preprocessing bits, that are stored by the server.
- *Message complexity*: for each query, the number of anonymous messages required to send. This is separately considered from the communication cost, since we need to take into account the anonymity cost. In particular, this will help us delineate between, e.g., sending one anonymous message of size s and sending s anonymous messages each of size 1 (since the latter may have more network overhead).

While our main focus is the server and the anonymity cost, we may also consider *per-query client computation*, which is the computational complexity for issuing each query and reconstructing the answer. One may also consider *client storage* which is omitted in this work as the clients in our constructions are stateless.

Warm-up impossibility result. When considering PIR with multiple clients, it is useful to study the minimum number of clients required for security. We show that for any *linear* PIR (i.e., its encoding function is linear), which includes the constructions mentioned in Section 4.1 and others [9, 15], the number of clients required is at least the database size. We also show that no linear PIR protocol in the shuffle model has statistical security better than $\frac{n-C}{n-1}$ for $C < n$. See details in the full version.

6 General Constructions for Single-Server Shuffle PIR

We now present generic ways to build asymptotically efficient PIR protocols in the shuffle model from standard multi-server PIR constructions. The high-level idea is to compose together a protocol OPIR at the *outer* layer with a protocol IPIR at the *inner* layer, for randomizing the query indices. We call this the *inner-outer* paradigm for ShPIR.

Motivating the inner-outer paradigm. Recall that following the split-and-mix technique, the analysis of [8, 23] directly implies a shuffle PIR protocol with 4 additive shares and $O(n^{3/4})$ communication. We find that using 2 additive shares (which would give $O(n^{1/2})$ communication) are not sufficient for two reasons: (1) client queries are not individually random; and (2) client queries may be arbitrarily correlated with each other. For example, it is easy to distinguish between sets of client indices that are far apart (e.g. all querying for index i vs index i'; see the full version for details), even if extra *noise* queries are added by the clients to reduce the statistical distance. Furthermore, if the queries are uniformly random (even if not independent), three additive shares are enough [8] although two shares are still not sufficient here. This motivates our two-layer approach below.

The insight of having OPIR. The key insight we use to navigate around this is to first *randomize* the query indices by using a separate *outer* PIR, which we denote as OPIR. The goal of this OPIR protocol is two fold: first, it reduces the distance between the two multi-sets I and I'; and second, it transforms the queries in a way that makes them pairwise-independent which turns out to be sufficient for us to prove security. Concretely, the OPIR protocol takes two multi-sets I and I', who may differ by as much as $\delta = C$, and constructs two new (larger) query multi-sets J and J', whose difference is now proportional to $\sqrt{\delta}$, and whose elements are now pairwise-independent. Then J and J' will be used as query indices of an *inner* PIR with additive (or CNF) shares. In this way, the server sees the IPIR sub-queries as if they were generated from random (and pairwise independent) query indices.

ShPIR compilation. To compile the overall ShPIR protocol, the server will need encode the database x twice: once using OPIR and once using IPIR. More precisely, the server first sets up a database consisting of the answers to every possible OPIR sub-queries based on x: it defines a new database $x' = (x'_1, \ldots, x'_{n'})$ of size $n' = |\mathcal{Q}_{\mathsf{OPIR}}|$ where each entry x'_i is set to be $\mathsf{OPIR.Answer}(P_x, L_i)$ where L_i denotes the i-th element in the sorting of $\mathcal{Q}_{\mathsf{OPIR}}$. If $\mathsf{OPIR.Answer}$ is different for different servers, then a size kn' (where k is the number of OPIR servers) database can be used, which concatenates all the n'-sized databases where the ℓ-th

database is defined using $\mathsf{OPIR.Answer}^{\ell}(P_x, L_i)$; see Construction B.1 for details. Now x', from the perspective of IPIR, is the database to be taken into the setup algorithm, i.e., the server runs $\mathsf{IPIR.Setup}(x')$, and the setup for ShPIR is done.

To query an index $i \in [n]$, a client will first use OPIR to generate queries $q_1, \ldots, q_k$ which are each uniformly random in the space $\mathcal{Q}_{\mathsf{OPIR}}$. Each of these q_ℓ can now be treated as an index i'_ℓ of the database x', following which the client will use IPIR.Query to fetch the i'_ℓ-th entry in x' that corresponds to q_ℓ. As a result, the final sub-queries to be sent to the server (along with additional noise) are generated by the client running IPIR.Query on the indices i'_ℓ for $\ell \in [k]$. The full details of the composition are given as Construction B.1 (Appendix B).

6.1 Composition with an Additive Two-Server IPIR

We start with our generic composition which uses an IPIR with two additive shares. We provide an overview of the core proof here; the full details are given in the full version.

▶ **Theorem 5** (ShPIR Composition Theorem for additive IPIR). *Let Φ be any k-server t-private information-theoretic PIR scheme where $k > t > 2$; denote its sub-query space size by Q and its answer size by A. Let Ψ be 2-additive PIR defined in Construction A.1. Then, for any database size $n \in \mathbb{N}$, given any $\epsilon > 0$, there exists a constant c_0 such that for $C \geq (c_0 Q^5)/(k\epsilon^8)$, the construction $\mathsf{ShPIR}(\Phi, \Psi)$ is a (Π, C, ϵ)-secure PIR in the shuffle model where Π is uniform. Here, Q, k, ϵ, C may all be functions of n. Furthermore, when $Q = \widetilde{O}(n)$ and assuming one-time preprocessing, the construction has:*

- *per-query server computation $O(A \cdot k^{\frac{3}{2}} \cdot Q^{\frac{1}{2}})$,*
- *per-query client computation $O(A \cdot k^{\frac{3}{2}} \cdot Q^{\frac{1}{2}})$,*
- *per-query communication $O(A \cdot k^{\frac{3}{2}} \cdot Q^{\frac{1}{2}})$,*
- *server storage $\widetilde{O}(A \cdot k^{\frac{3}{2}} \cdot Q^{\frac{3}{2}})$.*

▶ Remark 6 (Reduced cost for homogeneous servers). If OPIR has different Answer algorithms for the servers, the ShPIR server needs to store k sub-databases, where for ℓ-th sub-database the server treats $q \in \mathcal{Q}_{\mathsf{OPIR}}$ as the ℓ-th share and stores the corresponding answers. If OPIR.Answer is the same for all k servers, then ShPIR server only needs to store one such sub-database; as a result, both the per-query server computation and communication will be $O(A \cdot k \cdot Q^{\frac{1}{2}})$, and the server storage will be $O(A \cdot Q^{\frac{3}{2}})$. The client computation will be $O(A \cdot k \cdot Q^{\frac{1}{2}})$. See details in the full version.

6.1.1 Proof Outline of Theorem 5

Basic background. Consider a client query index $i \in [n]$. Recall that our k-server OPIR will first encode i into the space $\mathcal{E}_{\mathsf{OPIR}}$ and then split it into k sub-queries in the space $\mathcal{Q}_{\mathsf{OPIR}}$. When composing with the IPIR, these k sub-queries will now be interpreted as IPIR query indices within the IPIR database of size $|\mathcal{Q}_{\mathsf{OPIR}}|$. Each of the k indices will now be encoded within the IPIR encoding space $\mathcal{E}_{\mathsf{IPIR}}$, and then split into 2 shares in the space $\mathcal{Q}_{\mathsf{IPIR}}$. Note that the space $\mathcal{Q}_{\mathsf{OPIR}}$ and $\mathcal{E}_{\mathsf{IPIR}}$ have the same size, which is the size of the IPIR database, and that $\mathcal{E}_{\mathsf{IPIR}} \subset \mathcal{Q}_{\mathsf{IPIR}}$. Going forward, for clarity, we keep using "sub-queries" for OPIR but use "shares" to mean the sub-queries for IPIR.

Given C clients, we will have kC total IPIR query indices encoded into $\mathcal{E}_{\mathsf{IPIR}}$; denote this by $\mathsf{y} = (y_1, \ldots, y_{kC})$ and let $\widetilde{\mathsf{y}}$ (of length $2kC$) denote its shares in $\mathcal{Q}_{\mathsf{IPIR}}$. Our main goal is to analyze the properties of $\widetilde{\mathsf{y}}$ since this will be the view of the server. In particular, given two lists of original query indices $I = (i_1, \ldots, i_C)$ and $I' = (i'_1, \ldots, i'_C)$, and their resulting shares $\widetilde{\mathsf{y}}$ and $\widetilde{\mathsf{y}}'$, we want to understand whether an adversary can find e.g., which of I or I' corresponds to $\widetilde{\mathsf{y}}$.

Balls-and-bins-formulation. We now describe how to formulate our core analysis as a balls-and-bins problem. A key starting observation here is that a uniformly random shuffler Π will eliminate any ordering within $\widetilde{\mathsf{y}}$ (and similarly for y). In turn, this allows us to essentially do our analysis using a balls-and-bins formulation, where each share in $\widetilde{\mathsf{y}}$ corresponds to a ball in one of $|\mathcal{Q}_{\mathsf{IPIR}}|$ bins. More precisely, the distribution of the shuffled shares in $\widetilde{\mathsf{y}}$ is exactly a $|\mathcal{Q}_{\mathsf{IPIR}}|$-dimensional distribution where the each component represents the distribution of the number of balls in that bin. Towards this, we also find it helpful to analyze y using a similar balls-and-bins formulation.

The crux of our analysis now boils down to quantifying the statistical distance between the distribution of balls over bins resultant from any two sets of original query indices I and I'. Specifically, define $\mathcal{Y}(I)$ to be the distribution of the balls-and-bins configuration of IPIR query indices y resultant from the original query indices I; define $\widetilde{\mathcal{Y}}(I)$ to be the distribution of its shares (i.e., corresponding to $\widetilde{\mathsf{y}}$). Roughly, the goal now is to show that for any I and I', we can bound $\mathsf{SD}(\widetilde{\mathcal{Y}}(I), \widetilde{\mathcal{Y}}(I'))$ with some inverse polynomial in the number of clients.

Looking ahead however, we will require some extra balls to be added uniformly at random, essentially to "smooth out" the distribution of $\widetilde{\mathsf{y}}$; this can also be thought of as uniformly random *noise*. In the PIR context, this effectively corresponds to each client sending a random sub-query in $\mathcal{Q}_{\mathsf{IPIR}}$. We denote the balls-and-bins distribution of the shares with noise added as $\widetilde{\mathcal{Y}}^*(I)$.

▶ Remark 7 (Noise and communication complexity). We note that adding noise for each IPIR query index does not increase the asymptotic communication complexity for IPIR, i.e., the communication for an n-sized database is still $O(\sqrt{n})$. This is because the server will still evaluate each noise share either as the first or second share without changing the database encoding polynomial making the communication still $O(\sqrt{n})$. Note that adding noise is substantially different from splitting to more shares, i.e., if each IPIR index was instead split into more additive shares (corresponding to using an IPIR with more servers), then the number of variables in the encoding polynomial itself will be larger, which would increase the asymptotic communication.

Main proof steps. At a high level, we leverage balls-and-bins style analyses to bound the statistical distance between $\widetilde{\mathcal{Y}}^*(I)$ and $\widetilde{\mathcal{Y}}^*(I')$. The rough idea will be to first compute the *edit distance* between the balls-and-bins configurations corresponding to the IPIR shares and then use that to bound the statistical distance after adding the random noise. Our proof proceeds in three major steps which we outline below.

Proof Step 1: *(Analyzing the edit distance of* OPIR *sub-queries).* Consider two lists of client indices $I = (i_1, \ldots, i_C)$ and $I' = (i'_1, \ldots, i'_C)$. Abstractly, the first part of our proof shows that the edit distance between the OPIR sub-queries generated from I and I' is *not too large*.

Recall that the t-out-of-k OPIR sub-queries generated are individually uniformly random, and are $(t-1)$-wise independent (and therefore also pairwise independent). Therefore, we can formulate our objective as the following balls-and-bins problem given in Lemma 8.

▶ **Lemma 8.** *Suppose that B balls are thrown into N bins. Let $\mathcal{B}$ and $\mathcal{B}'$ be any two distributions of the final balls-and-bins configuration where each ball is thrown uniformly at random, and any two balls are independently thrown. Then* $\mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}[\mathsf{ED}(\mathbf{u},\mathbf{v})] \leq \sqrt{BN/2}$.

Casting this result to our construction, since each client index generates k OPIR sub-queries and there are C clients in total, the expectation of edit distance (or differences) between any two sets of OPIR sub-queries (and consequently, the IPIR indices) is at most $\sqrt{kC\,|\mathcal{Q}_{\mathsf{OPIR}}|/2}$.

Proof Step 2: *(Analyzing the edit distance of 2-additive sharing in the* IPIR*).* Now that we have a bound on the edit distance between OPIR sub-queries (and consequently IPIR indices), our next step is to analyze the edit distance for $\mathcal{Q}_{\mathsf{IPIR}}$ shares. Recall that each encoded $\mathcal{E}_{\mathsf{IPIR}}$ index is split into two additive shares. We model this as another balls-and-bins problem:

Consider a (B, N)-valid configuration $\mathbf{u}$ and let $\mathsf{Share}_{\mathbf{u}}$ denote the distribution of randomly splitting each ball in $\mathbf{u}$ (in a group $\mathbb{G}$), i.e., for each ball b, throw one ball into a random bin $u \leftarrow\!\!\$\, \mathbb{G}$, and another into bin $b - u$. The goal now is to bound the edit distance between $\mathsf{Share}_{\mathbf{u}}$ and $\mathsf{Share}_{\mathbf{v}}$ given the edit distance between $\mathbf{u}$ and $\mathbf{v}$.

To begin, we show that in the context of the final statistical distance, it is sufficient to only consider the parts of $\mathbf{u}$ and $\mathbf{v}$ that are different. Let $\mathsf{Share}^{\ell}_{\mathbf{u}}$ denote the distribution of the balls-and-bins configuration when further throwing ℓ balls independently and uniformly at random following the sharing $\mathsf{Share}_{\mathbf{u}}$. In particular, we show that,

$$\mathsf{SD}(\mathsf{Share}^{\ell}_{\mathbf{u}}, \mathsf{Share}^{\ell}_{\mathbf{v}}) \leq \mathsf{SD}(\mathsf{Share}^{\ell}_{\mathbf{u}\ominus\mathbf{v}}, \mathsf{Share}^{\ell}_{\mathbf{v}\ominus\mathbf{u}})$$

where $\ominus$ denotes the ball-difference operation defined in Section 4.2. Essentially, this will allow us to look at the splitting of only those balls that differ between $\mathbf{u}$ and $\mathbf{v}$; in particular, given (B, N)-valid $\mathbf{u}$ and $\mathbf{v}$ with edit distance δ, we will only need to concern ourselves with the (δ, N)-valid $\mathbf{u}' = \mathbf{u} \ominus \mathbf{v}$ and $\mathbf{v}' = \mathbf{v} \ominus \mathbf{u}$. We use this to show that $\mathbb{E}\left[\mathsf{ED}(\mathsf{Share}_{\mathbf{u}'}, \mathsf{Share}_{\mathbf{v}'})\right] \leq \sqrt{2\delta N}$. Combining this with the first part, we get:

$$\begin{aligned}\mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}\left[\mathsf{ED}(\mathsf{Share}_{\mathbf{u}\ominus\mathbf{v}}, \mathsf{Share}_{\mathbf{v}\ominus\mathbf{u}})\right] &\leq \sqrt{2N} \cdot \mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}\left[\sqrt{\mathsf{ED}(\mathbf{u},\mathbf{v})}\right] \\ &\leq \sqrt{2N}\,(BN/2)^{1/4} = (2)^{1/4}B^{1/4}(N)^{3/4}\end{aligned}$$

where the second step is by the concave Jensen's inequality.

Proof Step 3: *(Bounding the final statistical distance).* We are now ready to bound the final statistical distance between the final views of the server: $\widetilde{\mathcal{Y}}^*(I)$ and $\widetilde{\mathcal{Y}}^*(I')$. For this, we leverage a recent analysis by Boyle et al [11]. A straightforward corollary of their result can be abstractly stated as follows: Consider ℓ balls thrown independently and uniformly at random into N bins and let $\mathcal{U}_j$ denote the final distribution after another ball is added into bin j. Then for all bins j and j', we have $\mathsf{SD}(\mathcal{U}_j, \mathcal{U}_{j'}) \leq \sqrt{N/\ell}$. Informally, this can also be thought of as a "toy in sand" problem of being able to hide the location (bin j or bin j') of an initial ball (i.e., the toy) after throwing in N random balls as noise (i.e., the sand). The same analysis can be extended to show that if there are Δ initial balls, after which ℓ random balls are thrown, the statistical distance will be bounded by $\Delta \cdot \sqrt{N/\ell}$. In the context of our PIR analysis, intuitively, Δ will represent the edit distance between $\mathsf{Share}_{\mathbf{u}\ominus\mathbf{v}}$ and $\mathsf{Share}_{\mathbf{v}\ominus\mathbf{u}}$, while the ℓ extra balls will represent the additional "noise" IPIR queries made. Note that when using this balls-and-bins analysis, we need to account for the fact that the edit distance is a distribution in our case, rather than a fixed number; it is straightforward to do so by using standard first-moment techniques (since we have a bound on the expectation).

Casting these analyses back to our PIR context, first notice that $\widetilde{\mathcal{Y}}^*(I)$ is nothing but the distribution $\mathsf{Share}^{\ell}_{\mathbf{u}\sim\mathcal{B}(I)}$ where $\mathcal{B}(I)$ is the distribution of OPIR sub-queries resulting from the indices I. Looking ahead, we will use $\ell = kC$ uniformly random IPIR queries (i.e., k per client) as noise. A crucial point here is that the number of extra balls per client needs to be constant in C so that the individual communication complexity of each client does not depend on the how many clients are making queries. In fact, this also required our bound on the ED of the 2-additive sharing to be $o(\delta)$.

Combining the results from the previous parts, we show our main result:

$$\mathsf{SD}(\widetilde{\mathcal{Y}}^*(I), \widetilde{\mathcal{Y}}^*(I')) < \frac{3 \cdot N^{5/8}}{B^{1/8}} = \frac{3\,|\mathcal{Q}_{\mathsf{IPIR}}|^{5/8}}{(kC)^{1/8}}.$$

since $N = |\mathcal{Q}_{\mathsf{IPIR}}|$ bins (query-space) and $B = kC$ balls (total sub-queries).

A final task is bounding $|\mathcal{Q}_{\mathsf{IPIR}}|$ by Q (i.e., the size of OPIR sub-query space). The high-level idea here is that we let each IPIR database entry be A bits and consequently $|\mathcal{Q}_{\mathsf{IPIR}}|$ can be made $\widetilde{O}(Q)$. Now, assuming that there are $C = \Omega(n^{5+\nu}/k)$ client queries for some constant $\nu > 0$, the statistical distance can be bounded by some inverse polynomial $1/\mathsf{poly}(n)$ in n. More specifically, suppose that we wanted to bound the statistical distance by some inverse polynomial $\epsilon(n)$. Then, assuming at least $C(n) = \Omega(n^5/(k \cdot \epsilon^8))$ client queries, the statistical distance is bounded by ϵ. Consequently, the construction satisfies (Π, C, ϵ)-security in the shuffle model where Π is the uniform shuffler.

6.2 Reducing Communication using CNF Shares

In this section, we describe how to generalize the IPIR to use CNF shares instead of additive shares. The upshot is that it allows us to reduce the communication complexity of the resultant ShPIR protocol to $O(n^c)$ for any constant $c > 0$.

Construction outline. In a standard multi-server PIR, using s additive shares instead of 2 results in an increased communication cost of $O(n^{(s-1)/s})$ but this can be reduced to $O(n^{1/s})$ at the cost of a stronger non-collusion assumption using a CNF sharing where each server is given a different $s-1$ sized subset of the additive shares. We show that the same strategy in fact also works in our inner-outer paradigm by using an IPIR with CNF-shares (the composed protocol is given in Figure A.2). This compilation is particularly interesting since it requires *no extra non-collusion assumptions* to get the gain in efficiency (since the shuffle model already consists only of a single server). Instead, the trade-off will arise in the minimum number of clients required for security.

▶ **Theorem 9** (ShPIR Composition Theorem for CNF IPIR). *Let Φ be any k-server t-private information-theoretic PIR scheme where $k > t > 2$; denote its sub-query space size by Q and its answer size by A. Let Ψ be the s-CNF PIR defined in Construction A.2. Then, for any database size $n \in \mathbb{N}$, and given any $\epsilon > 0$, there exists a constant c_0 such that for $C \geq (c_0 Q^{2s+1})/(k\epsilon^8)$, the construction $\mathsf{ShPIR}(\Phi, \Psi)$ is a (Π, C, ϵ)-secure PIR in the shuffle model where Π is uniform. Here, Q, k, ϵ, C may all be functions of n. Furthermore, when $Q = \widetilde{O}(n)$ and assuming one-time preprocessing, the construction has:*

- *per-query server computation $O(A \cdot k^{1+1/s} \cdot Q^{1/s})$,*
- *per-query client computation $O(A \cdot k^{1+1/s} \cdot Q^{1/s})$,*
- *per-query communication $O(A \cdot k^{1+1/s} \cdot Q^{1/s})$,*
- *server storage $\widetilde{O}(A \cdot k^{1+1/s} \cdot Q^{1+1/s})$,*

Similar to Remark 6, if OPIR.Answer *is the same for all k servers, then both the per-query server computation and communication will be $O(A \cdot k \cdot Q^{1/s})$, and the server storage will be $O(A \cdot Q^{1+1/s})$. The client computation will be $O(A \cdot k \cdot Q^{1/s})$.*

Proof outline. The overall structure of the proof is very similar to the one for an additive IPIR; the main difference being in the second proof part to analyze the balls-and-bins distribution after the IPIR sharing which now involves CNF shares instead of additive shares.

Let s-CNF-Share$_{\mathbf{u}}$ be the distribution of the balls-and-bins configuration upon sharing each ball in $\mathbf{u}$ into s CNF shares in $\mathbb{G}^{s-1}$. Now, given (δ, N)-valid configurations $\mathbf{u}$ and $\mathbf{v}$, we want to bound the edit distance between s-CNF-Share$_{\mathbf{u}}$ and s-CNF-Share$_{\mathbf{v}}$; Through a natural group theoretic formulation, this turns out essentially reduce to understanding the (cyclic rotational) symmetries of the CNF-sharing. Concretely, this allows us to show that:

$$\mathsf{ED}(s\text{-CNF-Share}_{\mathbf{u}}, s\text{-CNF-Share}_{\mathbf{v}}) \leq sN^{(s-1)/2}\sqrt{\delta}.$$

Notice how this bound asymptotically generalizes the one from the 2-additive IPIR construction. Once we have this bound, the rest of the security proof of proceeds in exactly the same way as the one for Add-ShPIR. The complete proof is given in the full version.

6.3 Concrete Constructions based on Reed-Muller Code

We can now concretely instantiate OPIR with the Reed-Muller PIR and IPIR with the CNF PIR to achieve our main result below.

▶ **Theorem 10.** *For every constant $0 < \gamma < 1$, there exists a Reed-Muller PIR Φ and a $(\lceil 2/\gamma \rceil)$-CNF PIR Ψ, such that on any database size $n \in \mathbb{N}$, given any $\epsilon > 0$, for all $C \geq c_0 n^{1+4/\gamma}/\epsilon^8$ where c_0 is some constant, the construction* ShPIR(Φ, Ψ) *is a (Π, C, ϵ)-secure PIR where Π is uniform. Furthermore, assuming one-time preprocessing, we get:*

- *per-query server computation $O(n^\gamma)$,*
- *per-query client computation $O(n^\gamma)$,*
- *per-query communication $O(n^\gamma)$,*
- *per-query message complexity $O(n^\gamma)$,*
- *server storage is $\widetilde{O}(n^{1+\gamma/2})$.*

We defer the full proof of Theorem 10 to the full version. One thing to note here is that the reduced communication per client with CNF shares comes at a price – to achieve the same level of security, we need a larger number of clients.

▶ Remark 11 (Sub-polynomial communication assuming super-polynomial number of clients). An interesting consequence of the CNF-based IPIR is that it also enables more efficient protocols in the shuffle model. Using a $(\log n)$-server CNF-based protocol as our IPIR, we can achieve communication of $O(\mathsf{polylog}(n))$ with the assumption that there are at least some super-polynomial $n^{O(\log n)}$ number of clients. This results in better asymptotic complexity than the best existing protocols [20] in the standard-model PIR which use a constant numbers of servers. Note that the shuffle model compilation means that still only one server is required for our protocol and therefore we do not require the non-collusion assumptions of the standard-model CNF-based PIR.

▶ Remark 12 (Negligible security with slightly sublinear communication). Our main result only achieves inverse-polynomial rather than negligible security error. We note that if one settles for *slightly sublinear* communication, there is a simple solution that achieves negligible security error and proceeds as follows. The server writes the n-bit database as an $m \times m$ matrix over $\mathbb{Z}_2$ where $m = \sqrt{n}$. Each client writes the column it is interested in as a unit vector $q \in \mathbb{Z}_2^m$. Assuming C clients query at the same time, where C is super-linear in n, each client splits the vector q into $k = O((m+\sigma)/\log C)$ additive shares, for security parameter $\sigma = \log^2 n$. For each query $q' \in \mathbb{Z}_2^m$, the server responds with $X \cdot q' \in \mathbb{Z}_2^m$. By the tight security analysis of the *additive* split-and-mix protocol [8, 23, 32], the security error is negligible in n, i.e., $\Theta(1/n^{\log n})$, and both the query and the answer are of size $k \cdot m = O(n/\log n)$.

6.4 Combining with Standard-Model PIR

Our shuffle PIR can be used as a blackbox to reduce server cost for standard single-server PIR by any constant factor (even $10\times$ is a concretely substantial improvement).

Take any standard single-server PIR scheme stdPIR and denote the shuffle PIR construction as ShPIR. The server organizes the size-n database as an $\ell \times (n/\ell)$ matrix where ℓ is a constant. The key idea here is to use stdPIR to retrieve a column and ShPIR to retrieve a row. The server treats each column as a database in ShPIR and runs $\mathsf{ShPIR.Setup}$ on it. The server stores the preprocessed results as lookup tables (hence n/ℓ tables in total).

Suppose a client wants to retrieve the entry at r-th row and c-th column. The client runs the query algorithm of ShPIR on index $r \in [\ell]$ and generates k sub-queries. Then the client sends k messages anonymously, where the j-th message consists of the j-th sub-query of ShPIR and a stdPIR query for index $c \in [n/\ell]$. On receiving each message, the server first processes the sub-query of ShPIR (essentially n/ℓ table lookup operations), which results in n/ℓ elements; then the server processes the stdPIR query on these n/ℓ elements.

Compared to running stdPIR on a size-n database, this technique reduces server computation by a factor of ℓ. And the ShPIR database size is ℓ, which neither requires too many clients nor incurs high anonymity cost. The tradeoff is that a client sends k messages in the stdPIR-ShPIR combination instead of one message when using stdPIR only.

6.5 Lower Bound on Security

We show that for shuffle PIR protocols constructed in the inner-outer paradigm, $1/\mathsf{poly}(n)$ statistical security is tight in the sense that negligible security cannot be achieved with polynomially many clients using the additive inner PIR. The proof is deferred to the full version. This result does not rule out the information-theoretic constructions with negligible error, in particular, an interesting open problem to consider is instantiating the inner PIR with the Reed-Muller construction.

▶ **Theorem 13** (Lower bound on security for ShPIR)**.** *Let Φ be any multi-server PIR scheme. Denote the number of possible vectors of sub-queries as K_Φ. Let Ψ be a constant-server additive PIR (Construction A.1). On any database size $n \in \mathbb{N}$, for all (Π, C, ϵ)-secure $\mathsf{ShPIR}(\Phi, \Psi)$ constructions where C, K_Φ and K_Ψ are all bounded by polynomial $p_1(n)$, there exists a polynomial p_2 such that $\epsilon \geq 1/p_2(n)$.*

7 Conclusion and Open Questions

We demonstrate that PIR in the shuffle model can circumvent several limitations of standard-model PIR. This includes information-theoretic security with a single server, which opens a direction of constructing concretely efficient single-server schemes in the future.

The main technical question we leave open in this work is the possibility of obtaining similar results with negligible security error (recall that we can achieve this with slightly sublinear communication, see Remark 12). We conjecture that polylogarithmic communication per client with negligible security can be achieved by instantiating both OPIR and IPIR with the Reed-Muller PIR construction with a polylogarithmic security threshold and a polylogarithmic communication complexity.

Finally, an interesting direction for future research is obtaining concretely efficient PIR schemes in the shuffle model, possibly by settling for computational security.

References

1 Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder - scalable, robust anonymous committed broadcast. In *CCS*, pages 1233–1252, 2020.

2 Shweta Agrawal, Yuval Ishai, Eyal Kushilevitz, Varun Narayanan, Manoj Prabhakaran, Vinod M. Prabhakaran, and Alon Rosen. Secure computation from one-way noisy communication, or: Anti-correlation via anti-concentration. In *CRYPTO*, pages 124–154, 2021.

3 Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. In *PETS*, 2016.

4 Kinan Dak Albab, Rawane Issa, Mayank Varia, and Kalman Graffi. Batched differentially private information retrieval. In *USENIX Security*, pages 3327–3344, 2022.

5 Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication-computation trade-offs in PIR. In *USENIX Security*, pages 1811–1828, 2021.

6 Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *IEEE S&P*, pages 962–979, 2018.

7 Borja Balle, James Bell, and Adrià Gascón. Amplification by shuffling without shuffling. In *CCS*, pages 2292–2305, 2023.

8 Borja Balle, James Bell, Adria Gascon, and Kobbi Nissim. Private summation in the multi-message shuffle model. In *CCS*, pages 657–676, 2020.

9 Amos Beimel, Yuval Ishai, and Eyal Kushilevitz. General constructions for information-theoretic private information retrieval. In *Journal of Computer and System Sciences*, 2005.

10 Andrea Bittau, Ulfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. PROCHLO: Strong privacy for analytics in the crowd. In *SOSP*, pages 441–459, 2017.

11 Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In *CRYPTO*, pages 121–151, 2022.

12 David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM (CACM)*, 1981.

13 Albert Cheu, Adam D. Smith, Jonathan R. Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In *EUROCRYPT*, pages 375–403, 2019.

14 Albert Cheu and Jonathan R. Ullman. The limits of pan privacy and shuffle privacy for learning and estimation. In *STOC*, pages 1081–1094, 2021.

15 Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, pages 41–50, 1995.

16 Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *STOC*, pages 364–369, 1986.

17 Alex Davidson, Gonçalo Pestana, and Sofía Celi. FrodoPIR: Simple, scalable, single-server private information retrieval. In *PETS*, 2023.

18 Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.

19 Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *PODS*, 2003.

20 Zeev Dvir and Sivakanth Gopi. 2-server pir with sub-polynomial communication. In *STOC*, 2015.

21 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

22 Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Shuang Song, Kunal Talwar, and Abhradeep Thakurta. Encode, shuffle, analyze privacy revisited: Formalizations and empirical evaluation. *CoRR*, abs/2001.03618, 2020. URL: `https://arxiv.org/abs/2001.03618`.

23 Badih Ghazi, Pasin Manurangsi, Rasmus Pagh, and Ameya Velingker. Private aggregation from fewer anonymous messages. In *EUROCRYPT*, 2020.

24 Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In *EUROCRYPT*, 2010.

25 Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *IEEE S&P*, 2019.

26 Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with Popcorn. In *NSDI*, 2016.

27 Daniel Günther, Maurice Heymann, Benny Pinkas, and Thomas Schneider. Gpu-accelerated pir with client-independent preprocessing for large-scale applications. In *USENIX Security*, 2022.

28 Ryan Henry. Polynomial batch codes for efficient IT-PIR. In *PETS*, 2016.

29 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.

30 Kyle Hogan, Sacha Servan-Schreiber, Zachary Newman, Ben Weintraub, Cristina Nita-Rotaru, and Srinivas Devadas. Shortor: Improving tor network latency via multi-hop overlay routing. In *IEEE S&P*, 2022.

31 Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *STOC*, 2004.

32 Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *FOCS*, 2006.

33 Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing schemes realizing general access structure. In *IEEE Global Telecommunication Conference*, 1987.

34 Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *CCS*, 2016.

35 Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-go-mixes providing probabilistic anonymity in an open system. In *Information Hiding*, 1998.

36 Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *FOCS*, 1997.

37 Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew K. Miller. Honeybadgermpc and asynchromix: Practical asynchronous MPC and its application to anonymous communication. In *CCS*, pages 887–903, 2019.

38 Samir Jordan Menon and David J. Wu. Spiral: Fast, high-rate single-server pir via fhe composition. In *IEEE S&P*, 2022.

39 Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. In *TCC*, 2009.

40 Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-cost ϵ-private information retrieval. In *PETS*, 2016.

41 Jelle van den Hoof, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *SOSP*, 2015.

42 Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security*, 2016.

A Background on Multi-Server PIR Constructions

A.1 Two-Server PIR with Additive Shares

The first construction we describe is a PIR scheme from Beimel et al. [9] which uses two non-colluding servers. Figure A.1 contains the full description.

Setup. Consider a field $\mathbb{F}$ within which Σ can be encoded. The Setup algorithm encodes a database $x \in \Sigma^n$ into an m-variate polynomial $P_x \in \mathbb{F}[Z_1, \ldots, Z_m]$ as follows. First, choose m and $d < m$ such that $\binom{m}{d} \geq n$, and let $M = (M_1, \ldots M_n)$ denote a list of n monomials in the variables $Z_1, \ldots, Z_m$ with total degree exactly d and the degree of each variable at most

Let x be a database with size n and $\mathbb{F}$ be a field, where each entry x_i is in $\Sigma = \mathbb{F}$.

- $\mathsf{PIR.Setup}(x) \to P$:
  1. Choose m, d such that $\binom{m}{d} \geq n$.
  2. Let $M = (M_1, \ldots, M_n)$ be a list of n monomials in $\mathbb{F}[Z_1, \ldots, Z_m]$ with total degree d and intermediate degree at most 1. Sort all monomials that have m variables with degree d by a lexicographic order of the variables indices.
  3. Compute $P_x = \sum_{i=1}^{n} x_i M_i \in \mathbb{F}[Z_1, \ldots, Z_m]$.
  4. Compute a $2m$-variate degree-d polynomial P from P_x such that $P(Z_{1,1}, Z_{1,2}, \ldots, Z_{m,1}, Z_{m,2}) = P_x(Z_{1,1} + Z_{1,2}, \ldots, Z_{m,1} + Z_{m,2})$.
  5. Output P.
- $\mathsf{PIR.Query}(i; n) \to ((q_1, q_2), \mathsf{st})$, where $i \in [n]$:
  1. Let $\mathbf{z} = (z_1, \ldots, z_m)$ be the i-th binary vector such that $z_j = 1$ if and only if the monomial M_i contains the variable Z_j.
  2. Let $\mathbf{z}_1 \xleftarrow{\$} \mathbb{F}_2^m$, $\mathbf{z}_2 \leftarrow \mathbf{z} - \mathbf{z}_1$; and let $q_\ell \leftarrow \mathbf{z}_\ell$ for $\ell = 1, 2$. Set $\mathsf{st} = (\mathbf{z}_1, \mathbf{z}_2)$.
  3. Output $((q_1, q_2), \mathsf{st})$.
- $\mathsf{PIR.Answer}^\ell(P, q_\ell) \to a_\ell$ (for $\ell = 1, 2$):
  1. Let $\{M'_j\}_{j \in [2^m n]}$ be all monomials where the number $Z_{\_,\ell}$ is at least half of the variables.
  2. Output $a_\ell \leftarrow \sum_{j \in [2^m n]} M'_j(q_\ell)$.
- $\mathsf{PIR.Recon}((a_1, a_2), \mathsf{st}) \to x_i$:
  1. Parse st as $(\mathbf{z}_1, \mathbf{z}_2)$.
  2. Compute $x_i \leftarrow a_1(\mathbf{z}_2) + a_2(\mathbf{z}_1)$ (note that a_1 and a_2 are polynomials).
  3. Output x_i.

Construction A.1 A two-server information-theoretic PIR [9].

1. For simplicity, we pick the first n such monomials in lexicographic order of the variable indices. The encoding P_x is now simply the linear combination $P_x = \sum_{i=1}^{n} x_i M_i$.[1]

Query. The Query algorithm starts by encoding the query index $i \in [n]$ into a binary vector $\mathbf{z}^{(i)} = (z_1^{(i)}, \ldots, z_m^{(i)}) \in \{0,1\}^m$ defined such that each $z_j^{(i)} = 1$ if and only if the monomial M_i contains the variable Z_j. Observe here that the Hamming weight of $\mathbf{z}^{(i)}$ is d since the monomials are also of degree d. Such encoding ensures that $P_x(\mathbf{z}_i) = x_i$. Then the sub-queries are generated by splitting $\mathbf{z}^{(i)}$ into two additive shares $\mathbf{z}_1^{(i)} = (z_{1,1}^{(i)}, \ldots, z_{m,1}^{(i)})$ and $\mathbf{z}_2^{(i)} = (z_{1,2}^{(i)}, \ldots, z_{m,2}^{(i)})$, i.e., $\mathbf{z}^{(i)} = \mathbf{z}_1^{(i)} + \mathbf{z}_2^{(i)}$. Here, $\mathbf{z}_\ell^{(i)}$ is sent to the ℓ-th server for $\ell = 1, 2$.

Answer. The Answer^ℓ algorithm run by the servers first views the database encoding P_x as a $2m$-variate polynomial P'_x defined as:

$$P'_x(Z_{1,1}, Z_{1,2}, \ldots, Z_{m,1}, Z_{m,2}) = P_x(Z_{1,1} + Z_{1,2}, \ldots, Z_{m,1} + Z_{m,2}).$$

Now, the ℓ^{th} server selects all the monomial terms in P'_x such that the number of $Z_{\_,\ell}$ (i.e., the variables where the second subscript is ℓ) is at least half of the variables in that term (in the exactly half case, the monomials are split between the two servers in a pre-determined way). Note that the total number of monomials in P'_x is $2^d \cdot n$, so there should be $2^{d-1} \cdot n$ monomials for each server. The ℓ-th server then evaluates its selected monomials at the

[1] One can choose a more complicated encoding in [9] (E1 encoding scheme) that allows better parameters, namely $\sum_{\ell=0}^{d} \binom{m}{\ell} \geq n$.

point $\mathbf{z}_\ell^{(i)}$ and responds with the sum as the answer a_ℓ (which is now a polynomial in the remaining m variables). Further, observe that each monomial in P'_x is of degree d, and so after the server evaluation, the answer polynomial a_ℓ will be of degree at most $d/2$.

Reconstruction. Finally, given answer polynomials a_1, a_2, the client evaluates a_1 at $\mathbf{z}_2^{(i)}$ and a_2 at $\mathbf{z}_1^{(i)}$, and sums up the evaluation results in $\mathbb{F}$ to get $P_x(\mathbf{z}^{(i)}) = x_i$.

Cost. The parameters m and d can be chosen to be both $\Theta(\log n)$ such that $\binom{m}{d} \geq n$. In this case, the query size is $O(\log n)$ (since m elements in $\mathbb{F}_2$ are sent to each server) and the answer size is $O(\sqrt{n})$ (since specifying an m-variate polynomial of degree $d/2$ requires $\binom{m}{d/2} = O(\sqrt{n})$ terms).

Let x be a database with size n, each entry x_i is in $\Sigma = \mathbb{F}$. There are s non-colluding servers.

- $\mathsf{PIR.Setup}(x) \to P$:
  1. Choose m, d such that $\binom{m}{d} \geq n$.
  2. Let $M = (M_1, \ldots, M_n)$ be a list of n monomials in $\mathbb{F}[Z_1, \ldots, Z_m]$ with total degree exactly d and intermediate degree at most 1. Sort all monomials that have m variables with degree d by a lexicographic order of the variables indices.
  3. Compute $P_x = \sum_{i=1}^{n} x_i M_i \in \mathbb{F}[Z_1, \ldots, Z_m]$.
  4. Compute a sm-variate degree-d polynomial P from P_x such that $P(Z_{1,1}, \ldots, Z_{1,s}, \ldots, Z_{m,1} \ldots Z_{m,s}) = P_x(Z_{1,1} + \ldots + Z_{1,s}, \ldots, Z_{m,1} + \ldots + Z_{m,s})$.
  5. Output P.
- $\mathsf{PIR.Query}(i; n) \to ((q_1, \ldots, q_s), \mathsf{st})$, where $i \in [n]$:
  1. Let $\mathbf{z} = (z_1, \ldots, z_m)$ be the i-th binary vector such that $z_j = 1$ if and only if the monomial M_i contains the variable Z_j.
  2. Let $\mathbf{z}_1, \ldots, \mathbf{z}_{s-1} \xleftarrow{\$} \mathbb{F}_2^m$ and $\mathbf{z}_s \leftarrow \mathbf{z} - \sum_{j=1}^{s-1} \mathbf{z}_j$.
  3. Let $q_\ell \leftarrow (\mathbf{z}_{\ell+1}, \ldots, \mathbf{z}_s, \mathbf{z}_1, \ldots, \mathbf{z}_{\ell-1})$ for $\ell \in [s]$. // cyclic shift
  4. Set $\mathsf{st} = (\mathbf{z}_1, \ldots, \mathbf{z}_s)$.
  5. Output $((q_1, \ldots, q_s), \mathsf{st})$.
- $\mathsf{PIR.Answer}^\ell(P, q_\ell) \to a$, for $\ell \in [s]$:
  1. Let $\{M'_j\}_{j \in [s^m n]}$ be all monomials pre-determined such that the number of $Z_{\_,\ell}$ is at most $1/s$ fraction.
  2. Output $a \leftarrow \sum_{j \in [s^d n]} M'_j(q_\ell)$.
- $\mathsf{PIR.Recon}((a_1, \ldots, a_\ell), \mathsf{st}) \to x_i$:
  1. Parse st as $(\mathbf{z}_1, \ldots, \mathbf{z}_s)$.
  2. Compute $x_i \leftarrow \sum_{\ell \in [s]} a_\ell(\mathbf{z}_1, \ldots, \mathbf{z}_\ell - 1, \mathbf{z}_{\ell+1}, \mathbf{z}_s)$.
  3. Output x_i.

Construction A.2 An s-server PIR with CNF shares [9]. Note that when $s = 2$, this is simply the 2-server additive PIR.

***k*-server PIR with additive shares.** The above protocol can also be generalized to k servers where the encoding $\mathbf{z}$ is now split into k additive shares. In this case, the servers express the m-variate degree-d polynomial P_x as km-variate degree-d polynomial P'_x. Let $\mathcal{Z}_\ell$ be the set of monomials such that for each monomial, there are more $Z_{\_,\ell}$ than $Z_{\_,\ell'}$ for any $\ell' \neq \ell$. The set $\mathcal{Z}_\ell$ is assigned to the ℓ-th server. Moreover, the monomials in P'_x but not in any of $\mathcal{Z}_{\_,\ell}$'s will be divided to k servers in a pre-determined way. To issue a query for index i, the client encodes it as before to a binary string $\mathbf{z} \in \mathbb{F}_2^m$, and then splits it to k additive shares over $\mathbb{F}_2^m$, denoted as $\mathbf{z}_1, \ldots, \mathbf{z}_k$. The client sends to the ℓ-th server the share $\mathbf{z}_\ell$, and the

server evaluates the assigned monomials using $\mathbf{z}_\ell$. The evaluation result is a polynomial of degree $(k-1)d/k$; this implies the answer size (which dominates the communication cost) is $O(n^{(k-1)/k})$.

Observe that using more additive shares gives worse efficiency but better privacy (since collusion between any $k-1$ servers can be tolerated). Efficiency can be significantly improved to $O(n^{1/k})$ using CNF shares [33] (instead of additive shares) where each server is now given a different $(k-1)$-sized subset of the additive shares. This is because the evaluation of P_x at $k-1$ shares results in an answer polynomial of degree at most $O(n^{1/k})$. The efficiency gain, however, comes at the cost of much stronger non-collusion assumption for PIR, namely that no two database servers can collude. Looking ahead, an interesting consequence of using the shuffle model is that our CNF-sharing based construction (Section 6.2) can significantly reduce communication *without* making any non-collusion assumptions on database servers (since there is only one database).

For simplicity, going forward, we will refer to the k-server PIR with additive shares as k-*additive* PIR and its CNF-variant as k-CNF PIR.

A.2 k-Server PIR with Shamir Shares

In this section, we describe the k-server t-private PIR that uses Shamir secret sharing from [9]. Full description is provided in Figure A.3. We also call this the Reed-Muller PIR as it is closely related to Reed-Muller code.

Let $x = (x_1, \ldots, x_n) \in \mathbb{F}^n$ be a database.

$\mathsf{PIR.Setup}(x) \to P_x$:

1. Choose parameters m, d, k, t such that $\binom{m+d}{d} \geq n$ and $|\mathbb{F}| > k > td$.
2. Compute $P_x = \sum_{i=1}^{n} x_i P^{(i)}(z_1, \ldots, z_m)$, where $P^{(i)}(\mathsf{PIR.Enc}(i)) = 1$ and $P^{(i)}(\mathsf{PIR.Enc}(j)) = 0$ for all $i, j \in [n]$ and $i \neq j$.
3. Output P_x.

$\mathsf{PIR.Query}(i; n) \to ((q_1, \ldots, q_k), \mathsf{st})$, where $i \in [n]$:

1. Run $\mathsf{PIR.Enc}(i)$ and gets $\mathbf{z} \in \mathbb{F}^m$.
2. Choose a set of degree-t random polynomials $R = (R_1, \ldots, R_m)$ such that $R(\mathbf{0}) = \mathbf{z}$.
3. For $\ell \in [k]$:
   - Randomly choose r_ℓ from $\mathbb{F}$.
   - Set $q_\ell \leftarrow Q(r_\ell)$. Note that each $q_\ell \in \mathbb{F}^m$.
4. Set $\mathsf{st} = (r_1, \ldots, r_k)$.
5. Output $((q_1, \ldots, q_k), \mathsf{st})$.

$\mathsf{PIR.Answer}(P_x, q) \to a$:

1. Compute $a \leftarrow P_x(a)$.
2. Output a.

$\mathsf{PIR.Recon}((a_1, \ldots, a_k), \mathsf{st}) \to x_i$:

1. Parse $\mathsf{st} = (r_1, \ldots, r_k)$.
2. Interpolate a degree-td univariate polynomial $R \circ P_x$ from $\{(r_\ell, a_\ell)\}_{\ell=1}^{k}$.
3. Output $x_i \leftarrow (R \circ P_x)(0)$.

Construction A.3 A k-server t-private PIR based on Reed-Muller code [9].

Setup. Consider a field $\mathbb{F}$ within which Σ can be encoded. The Setup algorithm encodes a database $x \in \Sigma^n$ into a polynomial $P_x \in \mathbb{F}[Z_1, \ldots, Z_m]$ as follows: First, choose m and d such that $\binom{m+d}{d} \geq n$ and $|\mathbb{F}| > k > td$ (typically m, d, t are chosen first and then k and the field size $|\mathbb{F}|$ are deteremined accordingly). Let $\alpha_0, \ldots, \alpha_d$ be distinct elements in $\mathbb{F}$ (note that $d < |\mathbb{F}|$). The index i is encoded to the i-th vector $\mathbf{z}^{(i)}$ of the form $(\alpha_{\lambda_1}, \ldots, \alpha_{\lambda_m}) \in \mathbb{F}^m$ where $\sum_{j=1}^{m} \lambda_j \leq d$. There exists a set of polynomials $P^{(i)}(z_1, \ldots, z_m)$ of degree at most d such that $P^{(i)}(\mathbf{z}^{(i)}) = 1$ and $P^{(i)}(\mathbf{z}^{(j)}) = 0$ for all $i, j \in [n]$ and $i \neq j$. The full details of this encoding and the construction of $P^{(i)}$'s are provided in [9, Appendix B].

Query. To generate the sub-queries, after encoding the index i to $\mathbf{z}^{(i)}$, the client first chooses m univariate polynomials $(R_1, \ldots, R_m) = R$ each of degree t such that $R(\mathbf{0}) = (R_1(0), \ldots, R_m(0)) = \mathbf{z}^{(i)}$. It then randomly picks $r_1, \ldots, r_k \in \mathbb{F}$ and computes the sub-query to be sent to the ℓ^{th} server as $q_\ell = R(r_\ell) \in \mathbb{F}^m$.

Answer. The Answer^ℓ algorithm evaluates P_x at q_ℓ and sends back $a_\ell = P_x(q_\ell)$. Note that the answer algorithm for this protocol is the same for all k servers.

Reconstruction. Finally, the Recon algorithm uses Lagrange interpolation on the points (r_1, a_1), …, (r_k, a_k) to compute a degree td polynomial $S = P_x \circ R$; the evaluation $S(0)$ will give the desired database entry x_i. This interpolation is possible when $k > td$ and $|\mathbb{F}| > k$.

Other notation. For a PIR protocol Φ, we use $\mathcal{E}_\Phi$ to denote the encoding space of all indices. We use $\mathcal{Q}_\Phi$ to denote the space of all possible sub-queries (note that $\mathcal{Q}_\Phi$ may not equal $\mathcal{E}_\Phi$). For example, in the two-server construction above, $\mathcal{E}_\Phi$ contains all binary strings with Hamming weight d, and the space $\mathcal{Q}_\Phi$ is $\mathbb{F}_2^m$, i.e, in this case $\mathcal{E}_\Phi \subset \mathcal{Q}_\Phi$.

B Composed PIR Construction

The complete composed PIR construction in the inner-outer paradigm is given in Construction B.1.

ShPIR Composition. A shuffle model PIR protocol $\mathsf{ShPIR}(\mathsf{OPIR}, \mathsf{IPIR})$ built using the inner-outer paradigm from a k-server OPIR, and a s-server IPIR is defined as follows:

- $\mathsf{ShPIR.Setup}(x) \to P$:
  1. Let $P_x \leftarrow \mathsf{OPIR.Setup}(x)$.
  2. Define a database x' of size n' as follows:
     - Let $n^* = |\mathcal{Q}_{\mathsf{OPIR}}|$ and let $L = (L_1, \ldots, L_{n^*})$ denote the sorting of the sub-query space $\mathcal{Q}_{\mathsf{OPIR}}$.
     - If the Answer algorithm is the same for all OPIR servers:
       For all $i \in [n^*]$, let $x'_i \leftarrow \mathsf{OPIR.Answer}(P_x, L_i)$.
       As a result, x' is of size $n' = n^*$.
     - If the Answer algorithm is different for the k OPIR servers:
       For $i \in [n^*], \ell \in [k]$: let $x'_{i+n' \cdot (\ell-1)} \leftarrow \mathsf{OPIR.Answer}^\ell(P_x, L_i)$.
       As a result, x' is of size $n' = kn^*$.
  3. Run $\mathsf{IPIR.Setup}(x')$ and output its result as P.

- $\mathsf{ShPIR.Query}(i; n) \to (q_1, \ldots, q_h)$, where $i \in [n]$ and $h = k(s+1)$:
  1. Initialize $(u_{\ell,j})_{\ell \in [k], j \in [s]}$.
  2. Let $(q'_1, \ldots, q'_k) \leftarrow\!\!\$\ \mathsf{OPIR.Query}(i; n)$.
  3. For $\ell \in [k]$,
     - If the Answer algorithm is the same for all k OPIR servers:
       Map q'_ℓ to the corresponding index $i'_\ell \in [n']$,
       i.e., $x_{i'_\ell} = \mathsf{OPIR.Answer}(P_x, q'_\ell)$.
     - If the Answer algorithm is different for the k OPIR servers:
       Map q'_ℓ to the corresponding index $i'_\ell \in [kn']$,
       i.e., $x_{i'_\ell} = \mathsf{OPIR.Answer}^\ell(P_x, q'_\ell)$.
     - Let $(\widetilde{q}_1, \ldots, \widetilde{q}_s) \leftarrow\!\!\$\ \mathsf{IPIR.Query}(i'_\ell; n')$.
     - Set $(u_{\ell,1}, \ldots, u_{\ell,s}) \leftarrow (\widetilde{q}_1, \ldots, \widetilde{q}_s)$.
  4. Let $(r_1, \ldots, r_k) \xleftarrow{\$} \mathcal{Q}_{\mathsf{OPIR}}$. // dummies
  5. Output $(u_{1,1}, \ldots, u_{k,s}, r_1, \ldots, r_k)$.

- $\mathsf{ShPIR.Answer}(P, q) \to a$:
  1. If IPIR has the same Answer algorithms for server, return $a = \mathsf{IPIR.Answer}(P, q)$; otherwise return
  $$a = \left\{(\mathsf{IPIR.Answer}^\ell(P, q), \mathsf{label}\ \ell)\right\}_{\ell \in [s]}.$$

- $\mathsf{ShPIR.Recon}(a_1, \ldots, a_h) \to x_i$:
  1. Initialize $(v_{\ell,j})_{\ell \in [k], j \in [s]}$ and $(a'_\ell)_{\ell \in [k]}$.
  2. For $\ell \in [k]$, $j \in [s]$:
     - Let $a_{(\ell-1)\cdot k+j}$ be the answer to sub-query $q_{(\ell-1)\cdot k+j}$, namely $u_{\ell,j}$.
     - If IPIR has different Answer algorithms for the servers, parse $a_{(\ell-1)\cdot k+j}$ as
     $$\{(\widetilde{a}_1, \mathsf{label}\ 1), \ldots, (\widetilde{a}_s, \mathsf{label}\ s)\}, \text{ let } v_{\ell,j} := \widetilde{a}_j \text{ (whose associated label is } j).$$
     - If IPIR has the same Answer algorithms for the servers, let $v_{\ell,j} = a_{(\ell-1)\cdot k+j}$.
  3. For $\ell \in [k]$:
     - $a'_\ell \leftarrow \mathsf{IPIR.Recon}(v_{\ell,1}, \ldots, v_{\ell,s})$.
  4. Output $x_i \leftarrow \mathsf{OPIR.Recon}(a'_1, \ldots, a'_k)$.

Construction B.1 Composed ShPIR built using the inner-outer paradigm.