Information-Theoretic Multi-Server Private Information Retrieval with Client Preprocessing*

Jaspal Singh
Purdue University & Georgia Institute of Technology

Yu Wei
Georgia Institute of Technology

Vassilis Zikas
Georgia Institute of Technology

May 22, 2026

Abstract

A private information retrieval (PIR) protocol allows a client to fetch any entry from single or multiple servers who hold a public database (of size n) while ensuring no server learns any information about the client's query. Initial works on PIR were focused on reducing the communication complexity of PIR schemes. However, standard PIR protocols are often impractical to use in applications involving large databases, due to its inherent large server-side computation complexity, that's at least linear in the database size. Hence, a line of research has focused on considering alternative PIR models that can achieve improved server complexity.

The model of private information retrieval with client preprocessing has received a lot of interest beginning with the work due to Corrigan-Gibbs and Kogan (Eurocrypt 2020). In this model, the client interacts with two servers in an offline phase and it stores a local state, which it uses in the online phase to perform PIR queries. Constructions in this model achieve online client/server computation and bandwidth that's sublinear in the database size, at the cost of a one-time expensive offline phase. Till date all known constructions in this model are based on symmetric key primitives or on stronger public key assumptions like Decisional Diffie-Hellman (DDH) and Learning with Error (LWE). This work initiates the study of unconditional PIR with client preprocessing—where we avoid using any cryptographic assumptions. We present a new PIR protocol for $2t$ servers (where $t \in [2, \log_2 n/2]$) with threshold 1, where client and server online computation is $\widetilde{\mathcal{O}}(\sqrt{n})$[1]—matching the computation costs of other works based on cryptographic assumptions. The client storage and online communication complexity are $\widetilde{\mathcal{O}}(n^{0.5+1/2t})$ and $\widetilde{\mathcal{O}}(n^{1/2})$ respectively. Compared to previous works our PIR with client preprocessing protocol also has a very concretely efficient client/server online computation phase—which is dominated by xor operations, compared to cryptographic operations that are orders of magnitude slower. As a building block for our construction, we introduce a new information-theoretic primitive called *privately multi-puncturable random set* (PMPRS), which might be of independent interest. This new primitive can be viewed as a generalization of privately puncturable pseudorandom set, which is the key cryptographic building block used in previous works on PIR with client preprocessing.

*Work done while the authors were at Purdue University

[1] the $\widetilde{\mathcal{O}}(.)$ notation hides poly log factors

1 Introduction

First introduced by Chor et al. [CKGS98], a private information retrieval (PIR) protocol allows a client to fetch any entry of a public database held by a single or multiple non-colluding servers. A line of works beginning with Chor et al. [CKGS98] have focused on reducing the communication complexity of PIR in the single and multiple server cases under various cryptographic assumptions [CMS99, BI01, BIKR02, Yek08, Efr09, DG16]. PIR has been employed as a useful building block for many cryptographic applications, including private contact discovery [DRRT18, HSW23], anonymous communication [MOT+11], and safe browsing [KCG21].

Standard PIR protocols however are generally inefficient when they are used for applications involving very large databases. One major factor contributing to the inefficiency of all known PIR schemes is the linear server computation complexity per query. This inefficiency is inherent in the standard PIR model in both the single and multi-server case and both the statistical and computational setting [BIM00]. Hence, a line of research has focused on considering alternative PIR models, that allow for sublinear server complexity per query, either in the worst case or in an amortized sense. These include models focused on batch PIR queries [BIM00, SWP09, IKOS04, LG15, HHG13, AS16, ACLS18] and PIR with preprocessing [BIM00, LP23, SACM21, ZPSZ23, HHCG+23, CGHK22]. Specifically, the PIR with client preprocessing model has garnered a lot of attention beginning with a work by Corrigan-Gibbs and Kogan [CGK20].

PIR with client preprocessing In this 2-server PIR model introduced by Corrigan-Gibbs and Kogan [CGK20], the client interacts with two non-colluding servers during an offline phase where the servers receive as input a database of size n. At the end of this phase the client maintains some sublinear sized state and there's no state stored on the server side. This offline phase is often computationally expensive - with each server doing linear computation in the database size. In the online phase, the client can make an unbounded number of PIR queries using its stored state - such that both online communication and online client/server computation are sublinear in the database size! In [CGK20] the authors were able to construct a PIR protocol in this model with $\widetilde{\mathcal{O}}(n^{1/2})$ online client/server computation and $\widetilde{\mathcal{O}}\ (n^{1/2})$ client state size. Furthermore, in the online phase, the client query size is $\widetilde{\mathcal{O}}\ (n^{1/2})$ and the server response is $\mathcal{O}(1)$ - leading to online communication complexity of $\widetilde{\mathcal{O}}(n^{1/2})$. Their original 2-server construction is based on one-way functions (OWF), but since then it's also been extended to the single server model and it's been improved using other cryptographic primitives [LP23, GZS24, SACM21, ZPSZ23, MSR23, FLLP24, HHCG+23, CGHK22]. The key cryptographic building block used in [CGK20] and follow-up works is some variant of *privately puncturable pseudo-random set*, which we describe in greater detail next. We refer to all PIR preprocessing protocols that are based on this primitive to be designed in the *Corrigan-Gibbs and Kogan (CGK) paradigm*.

Corrigan-Gibbs and Kogan (CGK) Paradigm (based on privately puncturable pseudo-random sets) A privately puncturable pseudo-random set consists of four algorithms (Gen, Set, Test, Punc). Gen is a randomized function that outputs a short key k corresponding to a pseudo-random set. Function $\mathsf{Set}(k)$ outputs the corresponding pseudo-random set, which has distribution computationally indistinguishable from a random set of size $\sqrt{n}$ from domain $[n] = \{0, 1, \ldots, n-1\}$. Function $\mathsf{Test}(k, x)$ outputs a bit checking whether $x \stackrel{?}{\in} \mathsf{Set}(k)$. $\mathsf{Punc}(k, x)$ outputs a punctured key k', such that $\mathsf{Set}(k') = \mathsf{Set}(k) \setminus \{x\}$ and the key k' hides x. The first construction for this primitive in [CGK20] was based on pseudo-random permutations - where the key and punctured key have sizes κ and $\widetilde{\mathcal{O}}(\sqrt{n})$ respectively, where κ is the private key security parameter. The computation

complexity of Test and Set algorithms are $\widetilde{\mathcal{O}}(1)$ and $\widetilde{\mathcal{O}}(\sqrt{n})$ respectively.

A very rough sketch of the PIR with client preprocessing scheme of [CGK20] is as follows: the client generates $T = \widetilde{\mathcal{O}}(\sqrt{n})$ privately puncturable pseudo-random set keys $(k_1, k_2, \ldots, k_T)$ and it sends them to the first server in the offline phase. This server responds back with hint bits $h_i = \oplus_{j \in \mathsf{Set}(k_i)} DB[j]$ where DB is a database of size n held by both parties. The client stores these T keys and the corresponding hint bits as its client state. In the online phase, the client receives as input some queries $x \in [n]$ and it finds a key k_i from its state such that, $\mathsf{Test}(k_i, x) = \text{TRUE}$. It then sends the punctured key $k' \leftarrow \mathsf{Punc}(k_i, x)$ to the second server - which responds with $r \leftarrow \oplus_{j \in \mathsf{Set}(k')} DB[j]$, which we refer to as the 'database xor bit' with respect to the key k'. The client can now compute $DB[x] = r \oplus h_i$, which is the expected output of the PIR online phase.

This is a simplified version of the original protocol in [CGK20], and the original construction has a few more features. Firstly, in the online phase, the client also interacts with the first server to replenish the key and hint bit that it used to compute $DB[x]$, and this ensures that the client state always contains T privately puncturable pseudo-random keys before each online PIR query. Secondly, note that the right server in the above simplified protocol always views a key of a set of size $(\sqrt{n} - 1)$ punctured at x - and hence its view is **not** independent of x. The authors further use privacy amplification techniques to avoid this kind of leakage.

All follow-up PIR with preprocessing works in this CGK paradigm use cryptographic assumptions and they focus on designing a more efficient privately puncturable pseudo-random set - where the keys have short description size, while they allow for efficient set membership testing, set enumeration and puncturing [LP23, GZS24, SACM21]. In all these works, the client and server online computation is dominated by $\widetilde{\mathcal{O}}(\sqrt{n})$ cryptographic operations (either based on OWF or public-key primitives) and their bandwidth and client storage complexity have a multiplicative factor of cryptographic key length as well.

Information-theoretic setting In this work we focus our attention on the feasibility of designing PIR with client preprocessing schemes with sublinear client state, online computation and communication in the information-theoretic setting i.e. with no cryptographic assumption. In particular, in the Corrigan-Gibbs and Kogan paradigm we investigate the feasibility of designing an information-theoretic analog of privately puncturable pseudo-random sets. A major challenge here is to represent a pseudo-random set of size $\sqrt{n}$ with a key of size $o(\sqrt{n})$, while still allowing for efficient set membership, set enumeration and private puncturing.

The key observation that helps in the design of this primitive is that the correctness of the PIR scheme only requires that for any $x \in [n]$, $Pr(x \in \mathsf{Set}(k)) = 1/\sqrt{n}$, where $k \leftarrow \mathsf{Gen}()$, and it's not required that $\mathsf{Set}(k)$ has the same distribution as a random set of size $\sqrt{n}$ (which is a stricter requirement). We exploit this observation in our design of a privately puncturable random set, where the keys simultaneously have sufficient randomness and structure to ensure sublinear size, while allowing for puncturing that hides the punctured element.

Outside theoretical interest, PIR protocols in the information-theoretic setting would be attractive from a practical viewpoint as well. The computation complexity in information-theoretic protocols is dominated by simpler algorithmic operations (like bit shift, xor, etc), which are generally faster than cryptographic operations in both private-key and public-key regime.

1.1 Our Contribution

Information-theoretic PIR with client preprocessing We initiate the study of the PIR with client preprocessing model in the information theoretic setting based on the CGK paradigm

[CGK20]. We propose a $2t$ server PIR with preprocessing protocol with corruption threshold 1[2], where the client maintains a state of size $\widetilde{\mathcal{O}}(n^{1/2+1/2t})$, with online client computation $\widetilde{\mathcal{O}}(\sqrt{n})$, online per server computation $\widetilde{\mathcal{O}}(\sqrt{n})$ and online communication $\widetilde{\mathcal{O}}(\sqrt{n})$.

In particular, setting $t = 2$, we get a 4-server PIR with preprocessing protocols with client storage $\widetilde{\mathcal{O}}(n^{3/4})$, client/server online computation/bandwidth $\widetilde{\mathcal{O}}(n^{1/2})$. Setting $t = \log(n)/2$, we get a $\log(n)$-server PIR with preprocessing protocols with $\widetilde{\mathcal{O}}(n^{1/2})$ client storage, client/server online computation and online bandwidth $\widetilde{\mathcal{O}}(n^{1/2})$ - where these costs match the original 2-server PIR with preprocessing construction of Corrigan-Gibbs and Kogan [CGK20] based on OWFs.

All the client/server computation costs reported here are in number of bit operations, and unlike other PIR protocols it does not have a multiplicative $O(\mathsf{poly}\log(\kappa))$ factor, where κ is the security parameter. The online communication of our scheme has no security parameter multiplicative factor either, which is the case for all previous PIR with preprocessing constructions in the CGK model.

Improving PIR communication complexity In Section 4.1 we slightly modify the above construction to reduce the online bandwidth, and the online server response bandwidth at the cost of doubling the number of servers. We achieve a $4t$ server PIR with preprocessing protocol with threshold 1 with online bandwidth $n^{1/2t+o(1)}$, where the online computation asymptotic complexities of the client/server stay sublinear in the database size.

New information-theoretic primitive The key building block in our PIR construction is a (t, n)-*privately multi-puncturable random set* (PMPRS), which has five algorithms (Gen, Set, Test, Punc, $\mathsf{DotProdEval}$). Similar to the analogous cryptographic primitive, $\mathsf{Gen}()$ outputs a key k, where $\mathsf{Set}(k)$ outputs a set of size $\sqrt{n}$ with domain $[n]$ and each element from the domain is contained in the set with probability $1/\sqrt{n}$. Function $\mathsf{Test}(k, x)$ checks if $x \stackrel{?}{\in} \mathsf{Set}(k)$. The multi-puncturing function $\mathsf{Punc}(k, x)$ outputs t tuples of the form $(k_i, S_i, \mathsf{ind}_i)$ for $i \in [t]$, where each punctured key k_i corresponds to set S_i, and correctness requires that sets $S_0, \ldots, S_{t-1}$ are pairwise disjoint, and their union equals $\mathsf{Set}(k) \setminus \{x\}$. We call this primitive a 'multi-puncturable' random set, since the partitioned set $S \setminus \{x\}$ is divided into t disjoint sets. Privacy of this scheme requires that each of these punctured keys are simulatable given just the parameters t, n, which implies that they hide the punctured element x. This is a generalization of the traditional privately puncturable set primitive in [CGK20], where Punc function outputs a single punctured key. The function $\mathsf{DotProdEval}(k_i, i, DB)$ exactly captures the server computation - which involves generating the partial punctured set and compute database xor bit wrt the input punctured key. However, instead of outputting a single bit, this algorithm outputs a vector $\vec{v_i}$, such that the idx_i^{th} bit has the expected result i.e. $\vec{v_i}[\mathsf{idx}] = \oplus_{j \in S_i} DB[j]$. This kind of correctness requirement in the CGK model was also first considered in TreePIR [LP23] - which is a 2-server PIR with preprocessing construction based on DDH assumption.

We propose an information-theoretic construction for (t, n)-PMPRS (for when $n^{1/2t}$ is an integer) where the key and each punctured key have sizes $\widetilde{\mathcal{O}}(tn^{1/2t})$. The running time of $\mathsf{Gen}, \mathsf{Test}, \mathsf{Set}$, $\mathsf{DotProdEval}$ are $\widetilde{\mathcal{O}}(tn^{1/2t})$, $\widetilde{\mathcal{O}}(1), \widetilde{\mathcal{O}}(\sqrt{n})$ and $\widetilde{\mathcal{O}}(\sqrt{n})$ respectively.

1.2 Technical Overview

We divide our technical overview in two parts, first we highlight the key ideas behind our PMPRS construction, and next we show how this primitive can be used to construct a multi-server PIR

[2]i.e. all but one server are honest

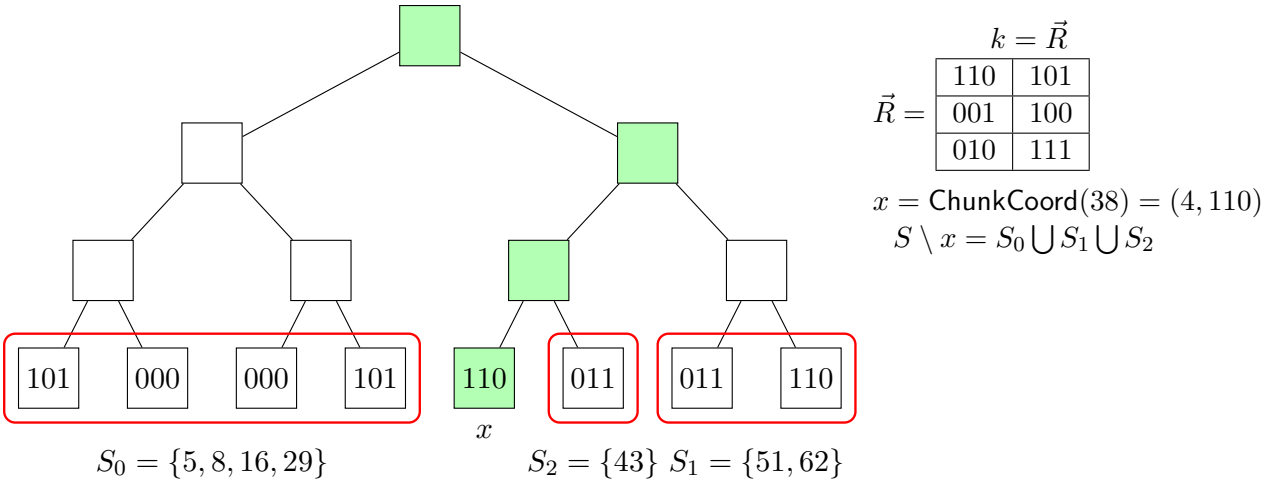

Figure 1: An example of $(t, n) - \mathsf{PMPRS}$ with $n = 64, t = 3$. The leaf nodes contain the offset values within the specific chunks, which can be computed using the vector $\vec{R}$. The sets S_0, S_1, S_2 represent the multi-puncturing obtained if the PMPRS set is punctured at $x = 38$ that's contained in the set. The leaf nodes corresponding to each set S_i and the path to the punctured element x are highlighted using the red boxes and the green filled boxes respectively

with client preprocessing.

$((\log_2 n)/2, n)$-PMPRS construction To illustrate some of the key ideas in our construction, in this subsection we depict a $((\log_2 n)/2, n)$-PMPRS construction where n is an even power of 2. The general (t, n) construction and its formal proof of security are presented in Section 3.

Our scheme generates PMPRS keys that correspond to *well-partitioned sets*, which are sets that contains a single element from each chunk of the domain $[n]$, where the ith chunk is defined as $\{i\sqrt{n}, i\sqrt{n}+1, \ldots, i\sqrt{n}+(\sqrt{n}-1)\}$. Hence, any well-partitioned set contains $\sqrt{n}$ elements - one for each chunk. We use the bijective map $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ to map any element $x \in [n]$ to its corresponding chunk $c_x = (\lfloor x/\sqrt{n} \rfloor) \in [\sqrt{n}]$ and the offset within the chunk $\delta_x =$ bit decomposition of $(x \mod \sqrt{n})$. We sometimes use the integer modulo $\sqrt{n}$ representation of δ_x as well, but it'll always be clear from the context. Sets with this structure were first used in a single server PIR with preprocessing construction PIANO [ZPSZ23], where the privately puncturable pseudo-random key is constructed using a pseudo-random function (PRF). The description of each algorithm in our PMPRS scheme is as follows:

- **Gen**(): outputs a matrix $\vec{R}$ of dimension $t \times 2$, where $t = (\log_2 n)/2$ and each element is sampled randomly from the domain $\{0, 1\}^t$. We use t as a shorthand for $\log_2(n)/2$ throughout the description of this construction.
- **Set**$(k = \vec{R})$: outputs a well-partitioned set, where the offset of the element in the ith chunk is given by $\oplus_{j=0}^{t-1} \vec{R}[j][i^j]$, where $(i^0, \ldots, i^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_2(i)$ is the bit decomposition of i. Since, $i \in [\sqrt{n}]$, the bit decomposition of i has $\log_2(\sqrt{n}) = t$ bits.
- **Test**$(\vec{R}, x)$: first compute $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$, and then check if the offset of the element in $\mathsf{Set}(\vec{R})$ in the c_x^{th} chunk is δ_x as follows: $\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j] \stackrel{?}{=} \delta_x$, where $(c_x^0, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_2(c_x)$.

We can visualize this well-partitioned set using a full binary tree $T_{2,n}$ with depth $t = \log_2(n)/2$ (and hence it has $\sqrt{n}$ leaves) as shows in Figure 1. We associate random values $(R[i][0], R[i][1])$

with depth i and we associate the ith leaf in the tree with the ith chunk. For any path from root to a leaf, we can xor one of the random strings at each depth, corresponding to whether the path travels along the left or the right child at that depth. Hence, the value computed at the ith leaf equals exactly the offset of the element in the ith chunk, as was computed in $\mathsf{Set}(\vec{R})$. We will use this tree based interpretation in the description of the following two algorithms of our PMPRS scheme:

- $((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1}) \leftarrow \mathbf{Punc}(k, x)$: To puncture the set $\mathsf{Set}(k)$ at element x, we can partition the tree $T_{2,n}$ after removing the path from root to the chunk containing x into t disjoint trees, where the ith tree T_i (for $i \in [t]$) has root at depth $i+1$, and it has 2^{t-i-1} leaves - which corresponds to the ith partitioned set S_i. Hence, we have $\cup_{i=0}^{t-1} S_i = S \setminus \{x\}$. An example of these t sets forming a disjoint union of the punctured set is also highlighted in Figure 1. However, note that each set S_i cannot be part of the key k_i since it leaks some information about x - particularly it leaks that this set doesn't contain the element of puncturing. To ensure privacy, while satisfying a correctness definition, we define the ith key k_i such that it contains sufficient information to compute offsets of all elements in S_i, but it contains no information about the chunks that correspond to those offsets in S_i. This decoupling of the offsets and the chunks is critical for making the scheme secure. Concretely, the key k_i has three components: a matrix $\vec{R}_i = \vec{R}[i+1:][:]$ i.e. R_i contains all rows $\geq i+1$ of R, it also contains a correction $\mathsf{corr} = \oplus_{j=0}^{i-1} \vec{R}[j][c_x^j]$ where $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and $(c_x^0, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_2(c_x)$, and finally it contains $R[i][1 - c_x^i]$. The first component of the key contains information from R from depth $i+1$ and higher, the second component contains partial information of R from depth 0 to $i-1$, and in particular it contains the xor of bits in R from these lower depths corresponding to the bit decomposition of c_x, and the third and final component contains one of the two random strings of R associated with depth i that is **not** used in computing δ_x (the offset of the punctured element). In the next function description we elaborate on how the indexes idx_i are computed and how the key k_i is used to construct S_i to satify the correctness definition. It's easy to see that key k_i hides the element x since it contains no information about $R[i][c_x^i]$ - masking the value of δ_x defined as $\oplus_{j=0}^{t-1} R[j][c_x^j]$, and it contains no information about the chunk containing the punctured element.
- $\vec{v}_i \leftarrow \mathbf{DotProdEval}(k_i, i, DB)$: Given $k_i = (\vec{R}_i, \mathsf{corr}, r)$, we first compute an offset vector $\vec{\delta}$ of length 2^{t-i-1} - such that this vector contains the offsets in the leaf nodes of tree T_i in increasing order of chunk indexes. This vector can be computed using $\vec{R}_i$ in a similar fashion to how the offset vector is computed in Set algorithm, and further each element of this is xored with $\mathsf{corr} \oplus r$. By construction, vector $\vec{\delta}$ contains offsets of all elements in S_i in order. However, note that k_i hides the exact chunk indexes (which depend on the item being punctured) that these offsets correspond to. Here, we make the key observation that there are exactly 2^{i+1} possible trees which could be T_i - one for each tree rooted at depth $(i+1)$ in $T_{2,n}$. Hence, for each of those 2^{i+1} trees in order we compute the following: consider the sequence of chunks (represented by a vector $\vec{c}$) corresponding to its leaf nodes in the tree, and compute a set $S' = \{\mathsf{ChunkCoord}^{-1}(\vec{c}[j], \vec{\delta}[j]) | j \in [2^{t-j-1}]\}$. Append to the vector v_i (which is initialized as null vector) with the bit $\oplus_{j \in S'} DB[j]$. Exactly one of these 2^{i+1} trees would be T_i, and let it be the idx^{th} tree in the sequence. Then by construction we have $\vec{v}_i[\mathsf{idx}_i] = \oplus_{j \in S_i} DB[j]$ - satisfying the correctness definition of PMPRS. It takes $O(2^{t-i-1})$ time to compute each bit of $\vec{v}_i$ and hence the running time of DotProdEval is $2^{i+1}.O(2^{t-i-1}) = O(2^t) = O(\sqrt{n})$.

This construction gives us an information theoretic PMPRS construction with key size $O(\log^2(n))$ for a $\sqrt{n}$ sized random set - such that it supports efficient set membership, set enumeration and

t-puncturing! We extend the above construction in Section 3 in a couple of ways. Firstly, we ensure that Gen can generate a set containing a specific element Δ (which is needed in the PIR construction), and secondly we give non-trivial PMPRS constructions for smaller t values. In the general construction we define $d = n^{1/2t}$ (which must be an integer) and we consider d-ary full tree $T_{n,d}$ (of depth $t = \log_d n$) over the domain $[n]$ instead of a binary tree. Here to puncture at a leaf node, we can partition the remaining tree into disjoint union of t "punctured trees" - which is defined as a tree with one of the root's children subtrees being removed. A major challenge in the general construction was to ensure that the $\mathsf{DotProdEval}$ has $O(\sqrt{n})$ complexity, as the trivial approach of considering all possible punctured subtrees at depth i leads to computation complexity $O(d\sqrt{n})$, which can be $\omega(\sqrt{n})$ for very small t or large d. We discuss this issue and the proposed solution in detail in Section 3.

$2t$-server PIR with client preprocessing Our PIR protocol follows the CGK paradigm. In the **offline phase** the client generates $T = \widetilde{O}(\sqrt{n})$ (t, n)-PMPRS keys and sends them to server 0. The server responds back with the hint bits for each of these keys, which is computed as follows for a given key k: $\oplus_{j \in \mathsf{Set}(k)} DB[j]$. The client stores the keys and the hint bits as its state.

In the **online phase**, the client inputs an index $x \in [n]$ and it finds a PMPRS key such that $\mathsf{Test}(k, x) = \text{TRUE}$. It computes $((k_0, \mathsf{idx}_0), \ldots, (k_{t-1}, \mathsf{idx}_{t-1})) \leftarrow \mathsf{Punc}(k, x)$[3], and sends k_i to server $(t + i)$. The server responds back with the vector $\vec{v_i} \leftarrow \mathsf{DotProdEval}(k_i, i, DB)$. By the correctness of the PMPRS primitive, we have $\oplus_{i=0}^{t-1} \vec{v_i}[\mathsf{idx}_i] = \oplus_{j \in \mathsf{Set}(S) \setminus \{x\}} DB[j]$. And hence if h is the hint bit corresponding to the key k, then the client can compute $h \oplus (\oplus_{i=0}^{t-1} \vec{v_i}[\mathsf{idx}_i]) = DB[x]$ - which is the desired output.

The above construction ends up using the pair (k, h) - and hence we replenish the state with a new key-hint pair to maintain the same client state. For this, the client samples a new key k' such that $x \in \mathsf{Set}(k')$. It punctures this key k' at x and it sends its t components to servers $0, 1, \ldots, (t-1)$ in the online phase. Each server responds back with vector output of $\mathsf{DotProdEval}$ algorithm. Using these vectors and the database bit $DB[x]$ the client can compute the hint bit $h' = \oplus_{j \in \mathsf{Set}(k')} DB[j]$.

The privacy of the scheme is ensured by the fact that in each online query each server only views a single punctured PMPRS key - which is simulatable by definition. And the correctness of this scheme follows from the definition of the PMPRS construction as described above. We defer the details of the security proof and the complexity analysis to Section 4.

1.3 Related Work

A trivial approach to solve the PIR problem would be for the client to download the entire database from the server, and store just the element of interest. However, this leads to linear bandwidth cost. Hence, a line of work starting with Chor et al. [CKGS98] have focused on reducing the bandwidth cost in the single server [KO97, CMS99, DGI+19, MW22, HHCG+23] and multi-server setting [BI01, BIKR02, Yek08, Efr09, GI14, BGI15, DG16]. However, all these works have linear computation complexity for each server - which is inherent in the standard PIR model as proven by Beimel et al. [BIM00]. To overcome this barrier, broadly speaking two models were introduced - PIR with batch queries and PIR with preprocessing.

In a PIR scheme with batch queries a client takes as input a sequence of k indexes, for which it privately queries the server(s). Here the goal is to amortize the server computation cost across the k queries. A number of works study this model of batch queries [BIM00, IKOS04, SWP09, HHG13,

[3]here we ignore the sets S_i output by Punc algorithm since they are not used in the PIR construction

LG15, AS16, ACLS18], and in particular the work due to Ishai, Kushilevitz, Ostrovsky and Sahai [IKOS04] achieve the optimal amortized per query server complexity of $\widetilde{\mathcal{O}}(n/k)$.

PIR with preprocessing was first proposed by Beimel, Ishai and Malkin [BIM00]. In their scheme, in the offline phase the two non-colluding servers do a one-time computation to store a new encoding of the database with super-linear size. In the online phase the server can support an unbounded number of client queries, where the client stores no state from the preprocessing phase. They introduce two information-theoretic protocols, one where each server stores a state of size $O(n^2)$ with online computation $O(n/\log^2 n)$, and bandwidth $O(n^{1/3})$. Their second scheme achieves online computation and bandwidth $O(n^{0.5+\epsilon})$ for any ϵ, where the server storage is $\omega(n)$ and it exponentially increases with decrease in ϵ. Compared to this information-theoretic preprocessing scheme, our construction has no super linear server state after preprocessing and it enjoys a lower client/server computation and bandwidth complexity at the price of a higher number of servers with corruption threshold 1.

PIR with client-side preprocessing was first introduced by Kogan and Corrigan-Gibbs [KCG21] in the 2-server model, which achieve client state, online client/server computation and bandwidth $\widetilde{\mathcal{O}}(\sqrt{n})$. This scheme was later improved in future works, where the focus is either to improve the asymptotic or concrete online bandwidth [LP23, GZS24, SACM21].Recently a number of single server PIR with preprocessing protocols were also proposed in the CGK paradigm [ZPSZ23, MSR23, FLLP24, HHCG+23, CGHK22] - where a single server can perform both the offline and online phase while satisfying the privacy requirement. All these previous works on PIR with client preprocessing are either in the OWF regime or they used some public key assumption like ϕ-hiding, DDH and LWE.

Concurrent Work A recent work due to Ishai et al. [ISW24] also study the problem of PIR with client preprocessing in the information-theoretic setting, but in the single and two server case. They provide a set of protocols with various performance tradeoffs between server computation, client space and bandwidth. In particular, all the proposed constructions with sublinear online server time in [ISW24] have client space, server computation and online bandwidth $\Omega(n^{2/3}), \Omega(n^{2/3})$ and $\Omega(n^{1/3})$ respectively. We improve on all these three performance metrics in the case of 4 or more servers in the information theoretic and client preprocessing PIR setting.

2 Preliminaries

2.1 Algorithmic Notation

A function $f : \mathbb{N} \to \mathbb{R}$ is called negligible it shrinks faster than any inverse polynomial i.e. for any polynomial $p()$, there exist an $N \in \mathbb{N}$, such that $f(n) < 1/p(n)$ for every $n \geq N$. We use the notation $\mathsf{negl}.(n)$ to represent any arbitrary negligible function in n. We use shorthand notation $S = \dot{\bigcup}_{i=0}^{m-1} S_i$ to represent that the m sets $S_0, \ldots, S_{m-1}$ are pairwise disjoint and their union equals set S.

Notation with an overset arrow (e.g. $\vec{v}, \vec{M}$) is used to represent vectors and matrices, where capitalized letters are used specifically for matrices. Notation $\leftarrow_{\$} R$ signifies sampling a random element from set R. For domain $[n] = \{0, 1, \ldots, n-1\}$, we define the ith chunk as the set $\{\sqrt{n}i, \sqrt{n}i + 1, \ldots, \sqrt{n}i + (\sqrt{n} - 1)\}$. Hence, we can view the domain $[n]$ as a disjoint union of $\sqrt{n}$ chunks. Define bijection $\mathsf{ChunkCoord}(x) = (c_x, \delta_x) \in [\sqrt{n}] \times \{0,1\}^{\log n/2}$, where $c_x = \lfloor x/\sqrt{n} \rfloor$ is the chunk that contains x and $\delta_x =$ bit decomposition of $(x \mod \sqrt{n})$ is the offset signifying which specific element in the chunk corresponds to x. We refer to c_x as the chunk coordinate of x. Sometimes in

the paper we refer to the $\mod \sqrt{n}$ representation of δ_x interchangeably with its bit decomposition - but it will always be clear from the context. A set S from domain $[n] = \{0, 1, \ldots, n-1\}$ is called *well partitioned* if it contains exactly one element from each chunk. Particularly, note that the description of a well partitioned set can be given by just a vector of offsets of size $[\sqrt{n}]$, which corresponds to offsets of the elements in each chunk.

Function $\vec{v} \leftarrow \mathsf{trim}(\vec{u}, i)$ takes as input a vector u (let say of size n) and an index $i \in [n]$, then it outputs a trimmed version of the input vector with the ith element removed. Hence, $\vec{v}[j] = \vec{u}[j]$ for $0 \leq j < i$ and $\vec{v}[j] = \vec{u}[j+1]$ for $j \in [i, n-1]$.

For any n, t where $d = n^{1/2t}$ is an integer, we use notation $T_{d,n}$ to represent a full d-ary tree of depth t $(= 1/2 \log_d n)$ - where the ith leaf node corresponds to the ith chunk of the domain $[n]$. Note $T_{d,n}$ has exactly $\sqrt{n}$ leaf nodes. We use the notation $\mathsf{chunks}_{d,n}(v)$ to represent increasing sequence of chunk indexes contained in subtree rooted at node v in tree $T_{d,n}$. Additionally, $\mathsf{chunks}_{d,n}(v, u)$ outputs the vector of increasing chunk indexes in sub-tree rooted at v in $T_{d,n}$ excluding the chunks/leaf nodes in the subtree rooted at the uth child of v, where $u \in [d]$. Hence, vector $\mathsf{chunks}_{d,n}(v, u)$ doesn't contain any chunk indexes contained in the subtree rooted at the uth child node of v.

2.2 Multi-Server PIR with client preprocessing (with threshold 1)

We adapt 2-server PIR with client preprocessing syntax and adaptive security definitions from Corrigan-Gibbs and Kogan [CGK20] and Shi et al. [SACM21] to the multi-server and the information-theoretic setting here.

An l server protocol contains $(l+1)$ parties: a single Client and l non-colluding servers Server_0, Server_1, $\ldots$, Server_{l-1}. All parties receive as input the statistical security parameter λ and the database size n. The protocol proceeds as follows:

- **Offline phase**: All the servers receive as input a database $DB \in \{0,1\}^n$. The client sends a single message to each server, which responds back with a single message to the client. The client uses these l responses to compute some state that it stores as output of this offline phase.
- **Online phase**: The servers can serve an unbounded queries of the following form: client receives as input an index $x \in [n]$, following which the client sends a single message to each of the servers as a function of its state and index x. Each server responds back to the client with a single message, which allows the client to compute an output bit $y \in \{0,1\}$.

Correctness For any database $DB \in \{0,1\}^n$ and an arbitrary sequence of queries $(x_1, x_2, \ldots)$, the client outputs $DB[x_i]$ at the end of the ith online query phase with probability at least $1 - \mathsf{negl}.(\lambda)$.

Privacy The PIR scheme is said to be private with threshold 1, if there exists a probabilistic polynomial time simulator $\mathsf{Sim}(1^\lambda, 1^n)$ such that for an adversary acting as the jth server (for any $j \in [l]$), polynomially bounded (in λ) parameters n and q and $DB \in \{0,1\}^n$, the view of the adversary $\mathcal{A}$ in the following two experiments is statistically indistinguishable:

- **Real**: An honest Client interacts with $\mathcal{A}(1^\lambda, 1^n, DB)$ who acts as Server_j and it may actively deviate from the prescribed PIR protocol. At the start of each online phase $i \in [q]$, $\mathcal{A}$ adaptively picks a query $x_i \in [n]$ which is the input of the Client in the same phase.
- **Ideal**: The simulator Sim acts as a Client and it interacts with $\mathcal{A}(1^\lambda, 1^n, DB)$ who acts as Server_j and which may actively deviate from the prescribed PIR protocol. At the start of each online phase $i \in [q]$, $\mathcal{A}$ adaptively picks a query $x_i \in [n]$ for the client, which is **not** input to Sim.

3 Privately multi-puncturable random set (PMPRS)

In this section we present formal syntax and security definition of our newly introduced PMPRS primitive. Following which we present our PMPRS construction, which is based on random sets with some structure imposed by d-ary trees (for $d = n^{1/2t}$) using a minimal amount of randomness.

Definition 1 (PMPRS syntax)**.** *A (t, n)-PMPRS scheme with input domain $[n] = \{0, \ldots, n-1\}$ consists of five algorithms (Gen, Set, Test, Punc, $\mathsf{DotProdEval}$) with the following syntax:*

- $k \leftarrow \mathsf{Gen}(\Delta, 1^t, 1^n)$*: outputs a short key $k \in \{0,1\}^*$ corresponding to a random set containing element $\Delta \in [n]$. The parameter Δ is an optional input to this algorithm*
- $S \leftarrow \mathsf{Set}(k)$*: takes as input a key k, and it outputs a random well partitioned set from domain $[n]$*
- $b \leftarrow \mathsf{Test}(k, x)$*: takes as input a key k, an element $x \in [n]$, and it outputs a boolean value* TRUE *or* FALSE*- corresponding to whether element x is contained in the set represented by k*
- $((S_0, k_0, \mathsf{ind}_0), (S_1, k_1, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(k, x)$*: outputs t punctured keys $k_0, \ldots, k_{t-1}$, with corresponding integer indexes $\mathsf{idx}_0, \ldots, \mathsf{idx}_{t-1}$ and sets $S_0, \ldots, S_{t-1}$, such that the t sets form a disjoint union of punctured set $\mathsf{Set}(k) \setminus \{x\}$.*
- $\vec{v}_i \leftarrow \mathsf{DotProdEval}(i, k_i, DB)$*: is a deterministic function that takes in a punctured key k_i, a vector $DB \in \{0,1\}^n$ and it outputs a vector $\vec{v}_i$, such that its ind_i^{th} bit corresponds to $\oplus_{j \in S_i} DB[j]$ - the database xor bit for one of the partitioned sets*

Definition 2 (PMPRS security)**.** *A (t, n)-PMPRS scheme (Gen, Set, Test, Punc, $\mathsf{DotProdEval}$) for domain $[n]$ is λ-**secure** if is satisfies the following conditions:*

- ***Correctness:*** *For any $\Delta, x \in [n]$ and $DB \in \{0,1\}^n$, let*

$$k \leftarrow \mathsf{Gen}(\Delta, 1^t, 1^n),\ S \leftarrow \mathsf{Set}(k)$$
$$((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(k, x)$$

  then the following holds:
  - $\Delta \in \mathsf{Set}(k)$
  - $S \setminus \{x\} = \bigcup S_i$
  - *for $i \in [t]$, $\vec{v}_i[\mathsf{ind}_i] = \oplus_{j \in S_i} DB[j]$*

  The second and third correctness requirements mentioned above also hold when the optional Δ parameter is not input to Gen algorithm
- ***Privacy:*** *There exists a simulator Sim such that for all $x \in [n], i \in [t]$, the following distributions are statistically indistinguishable in λ:*

$$\boxed{\begin{array}{l} k \leftarrow \mathsf{Gen}(x) \\ ((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(k, x) \\ \text{return } k_i \end{array}} \approx_\lambda \mathsf{Sim}(1^t, 1^n, i)$$

  Each punctured key k_i can be simulated using just the parameters t, n, i, or in other words, it hides the punctured element x. It should also be noted that the vectors $\vec{v}_i$ can be deterministically computed using the $\mathsf{DotProdEval}$ algorithm with input key k_i (which is simulatable) and vector DB. Hence, $\vec{v}_i$ hides element x as well, even if one of its bits correspond to the correct database xor bit on one of the punctured sets.
- ***Randomness:*** *The set output by $\mathsf{Set}(\mathsf{Gen}(1^t, 1^n))$ contains any element $x \in [n]$ with probability $1/\sqrt{n}$, where the probability is taken over the randomness of Gen algorithm. Additionally, $\mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))$ contains any element x not in the same chunk as Δ with probability $1/\sqrt{n}$*

Our PMPRS construction satisfies a stronger security guarantee which we define next.

Definition 3. *A λ-**secure** PMPRS scheme with $\lambda = 0$ is called perfectly secure.*

Efficiency requirements We measure the efficiency of any PMPRS scheme in terms of the size of the keys and the punctured keys - which would contribute to the communication potocol of our PIR scheme. We also measure the computation complexity of the Gen, Test, Set, Punc and DotProdEval algorithms, which would contribute to the computation complexity of the client and the servers in our PIR scheme.

3.1 Proposed PMPRS construction

We follow the blueprints of the PMPRS construction described in Section 1.2, but extend it to random sets generated using a d-ary tree structure instead of a binary tree. The formal description of our generic $(t, n)-$PMPRS construction, where $d = n^{1/2t}$ is an integer is given in Figure 3. We give a high level description of all the algorithms in this construction next.

The Gen function takes as input an additional Δ parameter, which should be contained in the random set corresponding to the output key k. The PMPRS key output of Gen consists of a matrix $\vec{R}$ of dimension $t \times d$ and an additional element corr. The value of corr is picked such that the well-partitioned set generated by k contains x.

The algorithm Set on input $(\vec{R}, \mathsf{corr})$ outputs a well partitioned set of size $\sqrt{n}$, where the element in chunk c with base d bit decomposition $(c^0, \ldots, c^{t-1})$ is given by $\mathsf{corr} \oplus \left(\oplus_{i=0}^{t-1} \vec{R}[i][c^i]\right)$. Intuitively, this refers to the xor of corr with the random strings in $\vec{R}$ corresponding to the path in tree $T_{d,n}$ from root to leaf c.

Hence, the algorithm Test on input x such that $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$, just checks if $\mathsf{Set}(k)$ has offset δ_x in chunk c_x. Note, that this doesn't require enumerating the entire well partitioned set, and it can be performed in time linear in the depth of the tree $T_{d,n}$.

Function Punc takes as input a PMPRS key k and the index of puncturing x, such that $x \in \mathsf{Set}(k)$. At a high level, this function outputs t punctured keys $k_0, \ldots, k_{t-1}$ and corresponding sets $S_0, \ldots, S_{t-1}$ such that the t sets form a disjoint union of punctured set $S \setminus \{x\}$. Removing the path from root to the leaf/chunk containing x in $T_{d,n}$ partitions the remaining tree into t "punctured trees", where the ith punctured tree (let's call it T'_i for $i \in [t]$) contains a subtree with root at depth i after *removing* the subtree rooted at exactly one of its children nodes. This structure is also depicted in Figure 2. The set S_i contains elements of S with chunk indexes exactly in the leaf nodes of sub-tree T'_i. Each key k_i is constructed such that it contains exactly the information needed to compute the offset (in order) of all elements corresponding to the leaf nodes in T'_i.

The function DotProdEval captures the computation performed by each server in our PIR scheme based on PMPRS. On input i, k_i, DB the goal of this algorithm is to compute the database xor bit of set S_i (i.e. $\oplus_{j \in S_i} DB[j]$). We can view this expected output as the dot product between two vectors: the database DB and the indicator vector $\vec{I}_{S_i}$ of set $S_i \subset [n]$[4]. However, our PMPRS scheme allows for a correctness notion - where DotProdEval outputs a vector $\vec{v}_i$ such that its idx^{th} bit (which was output of Punc) is the correct expected output. At a high level, this algorithm works in two stages:

- Given the punctured key k_i we can first compute an offset vector $\vec{\delta}$ - which contains the offsets of all elements in S_i in order from left to right chunk.

[4]the indicator vector $\vec{I}_S$ of a set S from domain [n] is a bit vector of size n such that $\vec{I}_S[i] = 1 \iff i \in S$

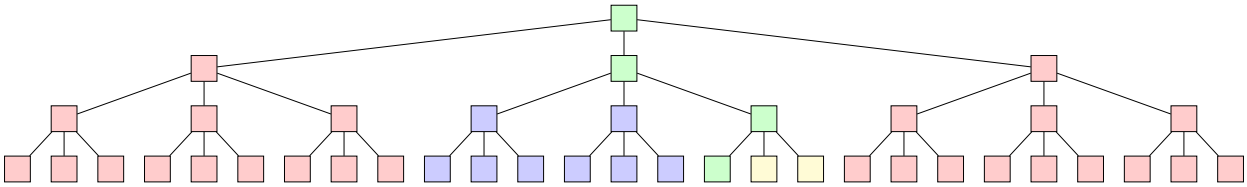

Figure 2: Example tree $T_{d,n}$ associated with $(t,n)-\mathsf{PMPRS}$ for parameters $n = 27^2, t = 3$, implying $d = 3$. The green path corresponds to the punctured element. Then punctured trees corresponding to sets S_0, S_1, S_2 output of Punc are colored red, blue and yellow respectively except for their roots, which are on the green path. Particularly note each of these "punctured trees" has root at a unique depth, and exactly one of their children subtrees missing

- Secondly, the algorithm computes the chunk vector $\vec{c}$ for every possible "punctured subtree" at depth i in $T_{d,n}$ - where exactly one of them is T'_i. For each of these possible punctured subtrees, we can compute the punctured set (given offsets $\vec{\delta}$ and corresponding chunk indexes $\vec{c}$). We use notation S_{uw} to represent corresponding to a tree rooted at node u with subtree at child node w punctured. The algorithm computes database xor bit for S_{uw} and it appends it to the output vector v_i. If idx_i refers to the index of chunk sequence for T'_i, then by construction $\vec{v}_i[\mathsf{idx}_i] = \oplus_{j \in S_i} DB[j]$ - proving the PMPRS scheme is correct. The privacy follows from the observation that the offset of the punctured element $x = \mathsf{ChunkCoord}^{-1}(c_x, \delta_x)$ is given by $\delta_x = \mathsf{corr} \oplus \left(\left(\oplus_{i=0}^{t-1} \vec{R}[i][c_x^i]\right)\right)$ where $(c_0^x, \ldots, c_{t-1}^x)$ is the base-d bit decomposition of c_x, and the fact that key k_i contains no information about $R[i][c_x^i]$ - which is one of the randomly sampled elements in Gen corresponding to depth i in tree $T_{d,n}$.

The trickiest part is to prove that $\mathsf{DotProdEval}$ has run time $\widetilde{\mathcal{O}}(\sqrt{n})$ on arbitrary input i, d_i, DB. Note that there are d^{i+1} punctured subtrees at depth i or sets S_{uw} that might correspond to the set S_i, since there are d^i nodes at depth i, where any of its d children subtrees could be punctured. Each of these sets S_{uw} has size $d^{t-i} - d^{t-i-1}$. Hence, trivially computing the database xor bit for each of these sets would lead to complexity $\widetilde{\mathcal{O}}(d^{t-i-1}(d-1)d^{i+1}) = \widetilde{\mathcal{O}}((d-1).d^t) = \mathcal{O}(d\sqrt{n})$, which can be $\omega(\sqrt{n})$ when $d = \omega(1)$. To reduce the computation complexity, we make the key observation that for any node u in $T_{d,n}$ at depth i and two adjacent children nodes w, w' of u, the sets S_{uw} and $S_{uw'}$ only differ in $2.d^{t-i-1}$ elements, and otherwise they overlap. Hence, given the database xor bit for set S_{uw}, we can compute the database xor bit for set $S_{uw'}$ in time $\widetilde{\mathcal{O}}(d^{t-i-1})$ instead of $\widetilde{\mathcal{O}}(d^{t-i})$ time that it takes to compute it trivially. This gives us the needed factor $O(d)$ improvement in the runtime - making the complexity of $\mathsf{DotProdEval}$ $\widetilde{\mathcal{O}}(\sqrt{n})$.

Theorem 4. *Let F be a (t,n)-PMPRS construction shown in Figure 3. Then F is perfectly secure.*

Proof. By Lemma 5, we know that F satisfies the correctness property defined in Definition 2. By Lemma 6, we know that F satisfies the randomness property defined in Definition 2. By Lemma 7, we can construct the simulator Sim that satisfies the following condition for $\lambda = 0$:

$$\boxed{\begin{array}{l} k \leftarrow \mathsf{Gen}(x) \\ ((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(Gen(\Delta, 1^t, 1^n), x) \\ \text{return } k_i \end{array}} \approx_\lambda \mathsf{Sim}(1^t, 1^n, i)$$

Our simulator construction works as follows: (on input t, n, i)

- Initialize $(d-1)$-length vector $\vec{r}_i$ where each element is uniformly distributed over $[\sqrt{n}]$.

Let $d = n^{1/2t}$

$\underline{\mathsf{Gen}\ (1^t, 1^n, \Delta)}$:

$(c_\Delta, \delta_\Delta) \leftarrow \mathsf{ChunkCoord}(\Delta)$

$(c_\Delta^0, c_\Delta^1, \ldots, c_\Delta^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_\Delta)$

Pick $t \times d$ random matrix $\vec{R}$ where each element is sampled $\leftarrow_\$ \{0,1\}^m$

If Δ parameter is input, set $\mathsf{corr} \leftarrow \delta_\Delta \oplus (\oplus_{i=0}^{t-1} \vec{R}[i][c_\Delta^i])$, else set $\mathsf{corr} \leftarrow_\$ \{0,1\}^m$

return $(\vec{R}, \mathsf{corr})$

$\underline{\mathsf{Set}(k = (\vec{R}, \mathsf{corr})) :}$

Initialize $S \leftarrow \{\}$

For each $c \in [\sqrt{n}]$:

$(c^0, c^1, \ldots, c^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c)$

update $S \leftarrow S \cup \left\{ \mathsf{ChunkCoord}^{-1}\left(c, \mathsf{corr} \oplus \left(\oplus_{i=0}^{t-1} \vec{R}[i][c^i]\right)\right)\right\}$

return S

$\underline{\mathsf{Test}(k = (\vec{R}, \mathsf{corr}), x) :}$

$(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$

$(c_x^0, c_x^1, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$

return $\mathsf{corr} \oplus \left(\oplus_{i=0}^{t-1} \vec{R}[i][c_x^i]\right) \stackrel{?}{=} \delta_x$

$\underline{\mathsf{Punc}(k = (\vec{R}, \mathsf{corr}), x)}$:

$(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$

$(c_x^0, c_x^1, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$

For $i \in [t]$:

Set $\vec{R}_i \leftarrow \vec{R}[i+1:][:]$

Set $\vec{r}_i \leftarrow \mathsf{trim}(\vec{R}[i], c_x^i)$

Set $\mathsf{corr}_i \leftarrow (\oplus_{j=0}^{i-1} \vec{R}[j][c_x^j]) \oplus \mathsf{corr}$

$k_i \leftarrow (\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$

Set $z \leftarrow \mathsf{bit\text{-}decomp}_d^{-1}(c_x^0, \ldots, c_x^{i-1})$

$\mathsf{ind}_i \leftarrow zd + c_x^i$

$u \leftarrow$ node in $T_{d,n}$ at depth i on path from root to c_x-th leaf node

$S_i \leftarrow$ subset of $\mathsf{Set}(k) \setminus \{x\}$ with chunk coordinates in $\mathsf{chunks}_{d,n}(u, c_x^i)$

return $((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1}))$

$\underline{\mathsf{DotProdEval}(i, k_i, DB)}$:

Parse k_i as $(\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$

Initialize vectors $\vec{v}_i, \vec{\delta}$ of size d^{i+1} and $(d-1) \cdot d^{t-i-1}$ respectively

For $j \in [d-1], j' \in [d^{t-i-1}]$:

$(c^0, c^1, \ldots, c^{t-i-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(j')$

$\vec{\delta}[j \cdot d^{t-i-1} + j'] \leftarrow \mathsf{corr} \oplus \vec{r}_i[j] \oplus \left(\oplus_{i'=0}^{i'=t-i-1} \vec{R}_i[i'][c^{i'}]\right)$

Initialize an iterator $j' \leftarrow 0$

For each depth i node u in $T_{d,n}$, its child node $w \in [d]$ (in left-to-right order):

$\vec{c} \leftarrow \mathsf{chunks}_{d,n}(u, w)$

Initialize set $S_{uw} \leftarrow \{\mathsf{ChunkCoord}^{-1}(\vec{c}[j], \vec{\delta}[j] |\ j \in [|\vec{\delta}|])\}$

Set $\vec{v}_i[j'] \leftarrow \oplus_{j \in S_{uw}} DB[j]$

Update $j' \leftarrow j' + 1$

Return $\vec{v}_i$

Figure 3: Proposed (t, n)-PMPRS construction (where $n^{1/2t}$ is an integer)

- Initialize $(t-i-1) \times d$ random matrix $\vec{R}_i$ where each element is uniformly distributed over $[\sqrt{n}]$.
- Initialize corr sampled from uniform distribution over $[\sqrt{n}]$.
- Return $k_i \leftarrow (\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$.

□

Lemma 5 (Correctness)**.** *The (t,n)-PMPRS construction shown in Figure 3 satisfies the **correctness** definition given in Definition 2.*

Proof. We first consider the case that the parameter Δ is given as input. First we check $\Delta \in \mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))$. This is by construction and we could show it passes the membership test $F.\mathrm{Test}(k, \Delta)$. Let $(c_\Delta, \delta_\Delta) \leftarrow \mathsf{ChunkCoord}(\Delta)$ and $(c_\Delta^0, c_\Delta^1, \ldots, c_\Delta^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_\Delta)$. We can verify that

$$\mathsf{corr} \oplus \left(\oplus_{j=0}^{t-1} R[j][c_\Delta^j]\right) = \delta_\Delta \oplus \left(\oplus_{j=0}^{t-1} R[j][c_\Delta^j]\right) \oplus \left(\oplus_{j=0}^{t-1} R[j][c_\Delta^j]\right) = \delta_\Delta.$$

Then we check $S \setminus \{x\} = \dot{\bigcup} S_i$. We first show that for distinct $i, j \in [t]$, S_i and S_j are disjoint. Let C_i and C_j be the corresponding set of chunk coordinates of S_i and S_j. We show S_i and S_j are disjoint by showing C_i and C_j are disjoint, this is simply because if two elements are in different chunks, they cannot be the same. WLOG, we consider the case $i < j$. Let $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and let p be the internal node in $T_{d,n}$ at depth i on path from root to c_x-th leaf node. By definition $C_i = \mathsf{chunks}_{d,n}(p, c_x^i)$, which is the set of chunks excluding the chunks/leaf nodes in the sub-tree rooted at c_x^i-th child of p and C_j is the subset of chunks in this excluded sub-tree, so C_i and C_j are disjoint. Then we show $S \setminus \{x\} = \bigcup S_i$. This is because, by construction, $\bigcup C_i = [\sqrt{n}] \setminus c_x$ includes all the leaf nodes of the tree $T_{d,n}$ except the c_x-th leaf, which represents the set $S \setminus \{x\}$.

We next show that for all $i \in [t]$, $v_i[\mathsf{ind}_i] = \oplus_{j \in S_i} DB[j]$. WLOG, we fixed an arbitrary $i \in [t]$. By ind_i's definition, we find the set S_{uw} where w is the c_x^i-th children node of u and u be the internal node in $T_{d,n}$ at depth i on the path from root to c_x-th leaf node. By checking the definition of S_{uw}, we can see it exactly equals to S_i. So we have

$$\vec{v}_i[\mathsf{ind}_i] = \oplus_{j \in S_{uw}} DB[j] = \oplus_{j \in S_i} DB[j].$$

Lastly, We consider the case that the optional parameter Δ is not given as input. We no longer have the requirement that $\Delta \in \mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))$. For the statement $S \setminus \{x\} = \dot{\bigcup} S_i$ and $v_i[\mathsf{ind}_i] = \oplus_{j \in S_i} DB[j]$, since our above proof doesn't rely on Δ, it still holds when Δ is not input. □

Lemma 6 (Randomness)**.** *Let F be a (t,n)-PMPRS construction shown in Figure 3. Then, for any $x \in [n]$,*

$$\Pr\left[x \in F.\mathsf{Set}(\mathsf{Gen}(1^t, 1^n))\right] = \frac{1}{\sqrt{n}}.$$

Additionally, for any $\Delta \in [n]$, and any $x \in [n]$ not in the same chunk as Δ,

$$\Pr\left[x \in F.\mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))\right] = \frac{1}{\sqrt{n}}.$$

Proof. Let $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and $(c_x^0, c_x^1, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$. Let X be the c_x-th element being added into the set $F.\mathsf{Set}(\mathsf{Gen}(1^t, 1^n)$. Then, we have

$$\begin{aligned}\Pr\left[x \in F.\mathsf{Set}(\mathsf{Gen}(1^t, 1^n))\right] &= \Pr\left[X = \delta_x\right] \\ &= \Pr\left[\mathsf{corr} \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right) = \delta_x\right] \\ &= \frac{1}{\sqrt{n}}.\end{aligned}$$

The second last step is by definition of Figure 3. The last step is because $\vec{R}[j][c_x^j]$ and corr are mutually independent and uniformly distributed over $[\sqrt{n}]$, so does the sum of them $\mathsf{corr} \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right)$.

Similarly, for any $\Delta \in [n]$, and any $x \in [n]$ not in the same chunk as Δ, let $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and $(c_x^0, c_x^1, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$. Let X be the c_x-th element being added into the set $F.\mathsf{Set}(\mathsf{Gen}(1^t, 1^n)$. let $(c_\Delta, \delta_\Delta) \leftarrow \mathsf{ChunkCoord}(\Delta)$ and $(c_\Delta^0, c_\Delta^1, \ldots, c_\Delta^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_\Delta)$. We have

$$\begin{aligned}\Pr\left[x \in F.\mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))\right] &= \Pr\left[X = \delta_x\right] \\ &= \Pr\left[\mathsf{corr} \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right) = \delta_x\right] \\ &= \Pr\left[\delta_\Delta \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_\Delta^j]\right) \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right) = \delta_x\right] \\ &= \Pr\left[\left(\oplus_{j=0}^{t-1} \vec{R}[j][c_\Delta^j]\right) \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right) = \delta_x \oplus \delta_\Delta\right] \\ &= \frac{1}{\sqrt{n}}.\end{aligned}$$

The last step is because every $\vec{R}[j][c_\Delta^j]$ and $\vec{R}[j][c_x^j]$ are mutually independent and uniformly distributed over $[\sqrt{n}]$ since $c_x \neq c_\Delta$ by definition. Therefore, the sum of them $\left(\oplus_{j=0}^{t-1} \vec{R}[j][c_\Delta^j]\right) \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right)$ is uniformly distributed over $[\sqrt{n}]$. □

Lemma 7 (Privacy)**.** *Let F be a (t, n)-PMPRS construction shown in Figure 3, $x \in [n]$, and $((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow F.\mathsf{Punc}(F.\mathsf{Gen}(x), x)$. For any $x \in [n], i \in [t]$, $k_i = (\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$ follows a joint distribution with size $(t - i) \times d$ where each component is independently and uniformly distributed over $[\sqrt{n}]$.*

Proof. We first show that every element in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$ follows a uniform distribution over $[m]$, and then we will show that elements in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$ are mutually independent.

Recall that $\vec{R}$ is a $t \times d$ random matrix where each element is uniformly distributed over $[\sqrt{n}]$. Let $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and $(c_x^0, c_x^1, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$. By corr's definition, $\mathsf{corr} = \delta_x \oplus (\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j])$ is uniformly distributed over $[\sqrt{n}]$. So $\mathsf{corr}_i \leftarrow (\oplus_{j=0}^{i-1} \vec{R}[j][c_x^j]) \oplus \mathsf{corr}$ is uniformly distributed over $[\sqrt{n}]$. Since all elements in $\vec{r}_i$ are defined as xor of elements in $\vec{R}$, so they are uniformly distributed over $[\sqrt{n}]$. Lastly, by $\vec{R}_i$'s definition, every element in $\vec{R}_i$ is a copy of an element in $\vec{R}$, so all elements in $\vec{R}_i$ are uniformly distributed over $[\sqrt{n}]$.

We now show elements in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$ are mutual independent.

- We initialize an empty set S and add all elements in $\vec{R}_i$ and $\vec{r}_i$ into S. We know that elements in S are mutual independent since, by definition, every element in $\vec{R}_i$ or $\vec{r}_i$ is a copy of an distinct element in $\vec{R}$ and so is independently sampled.

- By corr_i's definition, we know corr_i is independent of S, since $(\oplus_{j=0}^{i-1}\vec{R}[j][c_x^j])$ is independent of S.
- Update $S \leftarrow S \cup \{\mathsf{corr}_i\}$, we know all elements in S are mutual independent.

Since S equals the union of elements in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$, we conclude that elements in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$ are mutual independent. □

Theorem 8. *Let F be a (t,n)-PMPRS construction shown in Figure 3. Then*

- *The time complexity of $F.\mathsf{Test}(k,x)$ for any valid k,x is $O(\mathsf{poly}.\log(n))$.*
- *The time complexity of $F.\mathsf{Set}(k)$ for any valid k is $\widetilde{\mathcal{O}}(\sqrt{n})$.*
- *The time complexity of $F.\mathsf{Punc}(k,x)$ for any valid k,x is $\widetilde{\mathcal{O}}(\sqrt{n}+t^2n^{1/2t})$. Additionally, for $t \in [2, \frac{1}{2}\log(n)]$, $F.\mathsf{Punc}(k,x)$ runs in $\widetilde{\mathcal{O}}(\sqrt{n})$.*
- *The time complexity of $F.\mathsf{DotProdEval}(i,k_i,DB)$ for any valid i,k_i,DB is $\widetilde{\mathcal{O}}(\sqrt{n})$.*
- *The key k and punctured key k_i have size $\widetilde{\mathcal{O}}(tn^{1/2t})$ for every $i \in [t]$.*

Proof. We first note that for any input from $[n]$, all the $\mathsf{ChunkCoord}(\cdot)$, $\mathsf{ChunkCoord}^{-1}(\cdot,\cdot)$, $\mathsf{bit\text{-}decomp}(\cdot)$ and $\oplus$ operations can be done in $O(\mathsf{poly}.\log(n))$.

By $F.\mathsf{Test}(k,x)$'s definition, we can verify that it runs in $O(\mathsf{poly}.\log(n))$ for any valid k,x.

By $F.\mathsf{Set}(k)$'s definition, it runs a *for* loop $O(\sqrt{n})$ times and each loop can be done in $O(\mathsf{poly}.\log(n))$. Therefore, $F.\mathsf{Set}(k)$ runs in $O(\mathsf{poly}.\log(n)\cdot\sqrt{n}) = \widetilde{\mathcal{O}}(\sqrt{n})$ for any valid k.

By $F.\mathsf{Punc}(k,x)$'s definition, it computes $(S_i, k_i, \mathsf{ind}_i)$ for each server $i \in [t]$. k_i can be computed in $O(id\cdot\mathsf{poly}.\log(n))$, ind_i can be computed in $O(\mathsf{poly}.\log(n))$ and S_i can be computed in $O(d^{t-i}\mathsf{poly}.\log(n))$. Summing up together, $F.\mathsf{Punc}(k,x)$ runs in $O(\mathsf{poly}.\log(n)\cdot(\sqrt{n}+t^2n^{1/2t}))$. Additionally, if we choose $t \in [2,\frac{1}{2}\log(n)]$, $F.\mathsf{Punc}(k,x)$ runs in $O(\mathsf{poly}.\log(n)\cdot\sqrt{n}) = \widetilde{\mathcal{O}}(\sqrt{n})$.

We now compute the time complexity of $F.\mathsf{DotProdEval}(i,k_i,DB)$. To construct the offset vector $\vec{\delta}$, $\mathsf{DotProdEval}$ runs a *for* loop $O(d^{t-i})$ times and each loop can be done in $O(\mathsf{poly}.\log(n))$.

To compute a single element in $\vec{v}_i$, since the corresponding set S_{uw} is with size $O(d^{t-i})$, so the total computation time is $O(d^{t-i}\mathsf{poly}.\log(n))$. However, we note that, for adjacent element in $\vec{v}_i$, their corresponding set S_{uw} and $S_{uw'}$ are only different in $2\cdot d^{t-i-1}$ elements. This is because w and w' are sibling nodes and the chunks/leaf nodes in the sub-tree rooted at w and w' are the only elements that differentiate S_{uw} and $S_{uw'}$. This observation says that we only need to do a full computation on $\vec{v}_i[0]$, and all the rest elements of $\vec{v}_i$ can be computed in $O(d^{t-i-1}\mathsf{poly}.\log(n))$ time. Since $\vec{v}_i$ has d^{i+1} nodes, the total computation of $\vec{v}_i$ is $O(d^t\mathsf{poly}.\log(n))$.

Summing up the complexity of computing both vectors $\vec{\delta}$ and $\vec{v}_i$, we conclude that $F.\mathsf{DotProdEval}(i,k_i,DB)$ runs in $O(\mathsf{poly}.\log(n)\cdot d^t) = O(\mathsf{poly}.\log(n)\cdot\sqrt{n}) = \widetilde{\mathcal{O}}(\sqrt{n})$ for any valid i,k_i,DB.

Lastly, to see the size of $k = (\vec{R}, \mathsf{corr})$, since $\vec{R}$ is a $t\times d$ matrix where every entry has size $\frac{1}{2}\log_2 n$ and corr is a $\frac{1}{2}\log_2 n$-bits string. Therefore, k has size $O(td\cdot\mathsf{poly}.\log(n)) = \widetilde{\mathcal{O}}(tn^{1/2t})$. Similarly, by definition, punctured key $k_i = (\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$ has size $O(td\cdot\mathsf{poly}.\log(n)) = \widetilde{\mathcal{O}}(tn^{1/2t})$. □

4 Proposed scheme for PIR with client preprocessing

As described in the technical overview, our PIR construction is in the CGK paradigm, where instead of 2 servers, our construction assumes $2t$ servers for $t \geq 2$. Next we reiterate a high level outline for the PIR protocol, which we present formally in Figure 4.

Let n represent the size of the database DB, and $2t$ is the number of servers. In the *offline phase* the client generates PMPRS keys: $k_i \leftarrow \mathsf{Gen}()$ for $i \in [T] = \{1,2,,\ldots,T\}$, where $T = \lambda\sqrt{n}$ and λ is the statistical security parameter. The client sends these PMPRS keys to Server 0. Let $\vec{k}$ be a vector of keys such that $\vec{k}[i] = k_i$.

Server 0 interprets each of these PMPRS keys as a partitioned pseudo-random set (using Set function) each of size $\sqrt{n}$ - allowing it to compute the hint bit $h_i = \oplus_{j \in \mathsf{Set}(k_i)} DB[j]$. The Server 0 sends back the vector $\vec{h}$ with $\vec{h}[i] = h_i$ to the Client. The Client stores state $(\vec{k}, \vec{h})$ as output of the offline phase.

In the *online phase*, the client on input PIR index $x \in [n]$ first searches for a key k_i in $\vec{k}$ such that $\mathsf{Test}(k_i, x) = \text{TRUE}$. With probability $(1 - \mathsf{negl}.(\lambda))$ such a key would exist. This follows from the randomness property of the PMPRS primitive:

$$Pr(x \notin \mathsf{Set}(\mathsf{Gen}(1^t, 1^n))) = \left(1 - 1/\sqrt{n}\right)$$
$$\implies Pr\left(x \notin \mathsf{Set}(k_0) \wedge x \notin \mathsf{Set}(k_1) \ldots \wedge x \notin \mathsf{Set}(k_{T-1})\right) = (1 - /\sqrt{n})^{\lambda\sqrt{n}} \leq e^{-\lambda}$$

Next, the Client computes the punctured keys as $((S_0, k_0, \mathsf{ind}_0), (S_1, k_1, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(k, x)$ and sends to Server $(t + i)$ the punctured key k_i. Each of these servers respond back with vectors $\vec{v}_i$ as output by $\mathsf{DotProdEval}(k_i, i, DB)$. The client can now compute the PIR output $DB[x]$ as $h_i \oplus \left(\oplus_{j=0}^{t-1} \vec{v}_i[\mathsf{idx}_i]\right)$.

The client ends up consuming a PMPRS key and corresponding hint bit (k_i, h_i). To replenish the same, the Client generates a new PMPRS key k' containing x and it computes its hint bit by sharing punctured keys of $\mathsf{Punc}(k', x)$ with Servers $0, 1, \ldots t - 1$.

Theorem 9. *Suppose that F is ϵ_λ-secure (t, n)-PMPRS, then the $2t$-server PIR scheme (shown in Figure 4) that supports $\mathsf{poly}(\lambda)$ queries is ϵ_λ-private.*

Proof. The proof is a direct combination of Lemma 10 and Lemma 11. □

Lemma 10. *Suppose that F is ϵ_λ-secure (t, n)-PMPRS, then for any polynomial function $p(\cdot)$, and any adversary $\mathcal{A}$ that acts on behalf of Server $i \in \{t, \cdots, 2t-1\}$ and adaptively makes $p(\lambda)$ queries, there exists a PPT simulator $\mathsf{Sim}(1^\lambda, 1^n)$, where the polynomial is in terms of n, λ, such that*

$$\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}},$$

where $\text{view}_{Real}, \text{view}_{\mathsf{Sim}}$ are the distributions of $\mathcal{A}$'s views interacting with a real client in Figure 4 and Sim, respectively.

Proof. We first construct the following simulator Sim. Note that $\text{view}_{\mathsf{Sim}}$ follows the distribution $(k_i^1, \cdots, k_i^{p(\lambda)})$.

Simulator construction.
Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise computes the following:

- Ignores idx and samples a new index $y \leftarrow_\$ [n]$.
- $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, y)$.
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, y)$.
- Sends k_i^q to $\mathcal{A}$.

Indistinguishablity of view_{Real} and view_{SIM}.
To prove $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}}$, we follow a standard hybrid argument. We first construct Experiment Hyb1 described below. From the privacy property of of the underlying PMPRS scheme F, we have $\text{view}_{SIM} \approx_{\epsilon_\lambda} \text{view}_{Hyb1}$. We highlight the difference between Sim and Experiment Hyb1 with a shaded background.

Given a (t, n)-PMPRS F
Let $T = \lambda\sqrt{n}$.

Offline Phase:
Client computes $k_i \leftarrow F.\mathsf{Gen}(1^t, 1^n)$ for every $i \in [T]$
Client initializes vector $\vec{k}$ such that $\vec{k}[i] = k_i$
Client sends $\vec{k}$ to Server 0
Server 0 computes $h_i = \oplus_{j \in F.\mathsf{Set}(k_i)} DB[j]$ for every $i \in [T]$
Server 0 initializes vector $\vec{h}$ such that $\vec{h}[i] = h_i$ and send it back to the Client
Client stores $\vec{h}$ and $\vec{k}$ as its local state

Online Phase (Client inputs $x \in [n]$):
Client computes the following:
Find a $i \in [T]$ such that $F.\mathsf{Test}(\vec{k}[i], x) = \text{TRUE}$ and set $k \leftarrow \vec{k}[i]$ and $h \leftarrow \vec{h}[i]$
$((S_0, k_0, \mathsf{ind}_0), (S_1, k_1, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow F.\mathsf{Punc}(k, x)$
Client sends k_i to Server $t + i$ for each $i \in [t]$
Each server $t + i$ (for $i \in [t]$) computes the following:
$\vec{v}_i \leftarrow F.\mathsf{DotProdEval}(i, k_i, DB)$
Send $\vec{v}_i$ back to Client
Client on receiving $\vec{v}_i$ from each Server $t + i$ (for $i \in [t]$) computes:
$DB[x] \leftarrow \vec{v}_0[\mathsf{ind}_0] \oplus \cdots \oplus \vec{v}_{t-1}[\mathsf{ind}_{t-1}] \oplus h$
$k' \leftarrow F.\mathsf{Gen}(1^t, 1^n, x)$
$((S'_0, k'_0, \mathsf{ind}'_0), (S'_1, k'_1, \mathsf{ind}'_1), \ldots, (S'_{t-1}, k'_{t-1}, \mathsf{ind}'_{t-1})) \leftarrow F.\mathsf{Punc}(k', x)$
Client sends k'_i to Server i for each $i \in [t]$
Each server i (for $i \in [t]$) computes the following:
$\vec{v}'_i \leftarrow F.\mathsf{DotProdEval}(i, k'_i, DB)$
Send $\vec{v}'_i$ back to Client
Client on receiving $\vec{v}'_i$ from each Server i (for $i \in [t]$) computes:
$h' \leftarrow \vec{v}'_0[\mathsf{ind}'_0] \oplus \cdots \oplus \vec{v}'_{t-1}[\mathsf{ind}'_{t-1}] \oplus DB[x]$
Update the used (k, h) pair from vectors $\vec{k}, \vec{h}$ with (k', h')

Figure 4: Proposed $2t$ server PIR with preprocessing protocol for database of size n given a (t, n)-PMPRS

Experiment Hyb1 Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- $k_i^q \leftarrow \mathsf{Sim}_F(1^t, 1^n, i)$, where Sim_F is a simulator for F defined in Definition 2.
- Sends k_i^q to $\mathcal{A}$.

We again follow the Privacy property of the underlying PMPRS scheme F, and have $\text{view}_{Hyb1} \approx_{\epsilon_\lambda} \text{view}_{Hyb2}$.

Experiment Hyb2 Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Computes $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, idx)$.
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, idx)$.
- Sends k_i^q to $\mathcal{A}$.

We highlight the difference between the Realworld Construction and the Experiment Hyb2 with a shaded background. Note that view_{Real} follows the distribution $(k_i^1, \cdots, k_i^{p(\lambda)})$ in the Realworld Construction below. The difference between the Realworld Construction and the Experiment Hyb2 is that the puncturable random sets in *Real* is generated offline and there is a negligible probability of it not being able to find a random set containing idx (by the guarantee of choosing parameter T). Since the adversary didn't participate in the offline phase, it has no chance to see the puncturable random set generated in offline phase, so $\text{view}_{Hyb2} \approx_{\epsilon_\lambda} \text{view}_{Real}$.

Realworld Construction Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Find a $k \in \vec{k}$ such that $F.\mathsf{Test}(k, x) = \text{TRUE}$
- Set $k^q \leftarrow k$
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, idx)$.
- Sends k_i^q to $\mathcal{A}$.

By the standard hybrid argument, we conclude that $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}}$. □

Lemma 11. *Suppose that F is ϵ_λ-secure (t, n)-PMPRS, for any polynomial function $p(\cdot)$, and any adversary $\mathcal{A}$ that acts on behalf of Server $i \in [t]$ and adaptively makes $p(\lambda)$ queries, there exists a PPT simulator $\mathsf{Sim}(1^\lambda, 1^n)$, where the polynomial is in terms of n, λ, such that*

$$\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}},$$

where $\text{view}_{Real}, \text{view}_{\mathsf{Sim}}$ are the distributions of $\mathcal{A}$'s views interacting with a real client in Figure 4 and Sim, respectively.

Proof. We first construct the following simulator Sim for any adversary $\mathcal{A}$ that acts on behalf of Server $i \in \{1, \cdots, t-1\}$, and prove $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}}$. Then, we will show how to extend the simulator and the proof to $\mathcal{A}$ that acts on behalf of Server 0, which also participates the offline phase. Note that $\text{view}_{\mathsf{Sim}}$ follows the distribution $(k_i^1, \cdots, k_i^{p(\lambda)})$.

Simulator construction.

Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Ignores idx and samples a new index $y \leftarrow_{\$} [n]$.
- Computes $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, y)$.
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, y)$.
- Sends k_i^q to $\mathcal{A}$.

Indistinguishablity of view_{Real} and view_{SIM}. To prove $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}}$, we follow a standard hybrid argument. We construct Experiment Hyb1 in the below. Directly following the Privacy property of the underlying PMPRS scheme F, we have $\text{view}_{SIM} \approx_{\epsilon_\lambda} \text{view}_{Hyb1}$. We highlight the difference between Sim and Experiment Hyb1 with a shaded background.

Experiment Hyb1 Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- $k_i^q \leftarrow \mathsf{Sim}_F(1^t, 1^n, i)$, where Sim_F is a simulator for F defined in Definition 2.
- Sends k_i^q to $\mathcal{A}$.

We again follow the Privacy property of the underlying PMPRS scheme F, and have $\text{view}_{Hyb1} \approx_{\epsilon_\lambda} \text{view}_{Real}$.

Realworld Construction Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Computes $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, idx)$.
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, idx)$.
- Sends k_i^q to $\mathcal{A}$.

We then construct the simulator Sim_0 for any adversary $\mathcal{A}$ that acts on behalf of Server 0. The proof of $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}_0}$ follows exactly the same flow in the above.

Simulator construction.

In the offline phase,

- For $i = 1$ to T: computes $k_i \leftarrow F.\mathsf{Gen}(1^t, 1^n)$.
- Sends $k_0 \ldots, k_{T-1}$ to $\mathcal{A}$.

In the online phase, upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Ignores idx and samples a new index $y \leftarrow_{\$} [n]$.
- Computes $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, y)$.
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, y)$.
- Sends k_i^q to $\mathcal{A}$.

□

Theorem 12. *The $2t$-server PIR with client preprocessing protocol (in Figure 4) instantiated with the $(t, n) -$ PMPRS F (in Figure 3) has the following complexity:*

- $\widetilde{\mathcal{O}}(\lambda\sqrt{n}t^2 n^{\frac{1}{2t}})$ *client storage. If* $t \in [2, \log_2(n)]$, $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$ *client storage;*
- *No additional server storage after offline phase;*
- *Offline Phase:*
  - $\widetilde{\mathcal{O}}(\lambda n)$ *server time and* $\widetilde{\mathcal{O}}(\lambda\sqrt{n}tn^{\frac{1}{2n}})$ *client time; if* $t \in [2, \log_2(n)]$, $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$ *client time;*
  - $\widetilde{\mathcal{O}}(\lambda n^{1/2+1/2t})$ *communication;*
- *Online Phase:*
  - $\widetilde{\mathcal{O}}(\sqrt{n})$ *server time and* $\widetilde{\mathcal{O}}(\sqrt{n} + t^2 n^{1/2t})$ *client time; if* $t \in [2, \log_2(n)]$, $\widetilde{\mathcal{O}}(\sqrt{n})$ *client time;*

– $\widetilde{\mathcal{O}}(\sqrt{n}t)$ *communication; if* $t \in [2, \log_2(n)]$, $\widetilde{\mathcal{O}}(\sqrt{n})$ *communication.*

Therefore, the amortized communication per query is $\widetilde{\mathcal{O}}(\sqrt{n})$*, and the amortized server computation and client computation per query is* $\widetilde{\mathcal{O}}(\sqrt{n})$ *if we choose* $t \in [2, \log_2(n)]$.

Proof. On the client side, it stores the hint vector $\vec{h}$ and the key vector $\vec{k}$ and also needs a buffer to store $F.\mathsf{Punc}$'s output. Recall $\vec{h}$ and $\vec{k}$ both have size $T = O(\lambda\sqrt{n})$ and each element requires $O(1)$ and $\widetilde{\mathcal{O}}(tn^{\frac{1}{2t}})$ storage, separately. $F.\mathsf{Punc}$'s output requires $O(t\log(n))$, $\widetilde{\mathcal{O}}(t^2 n^{\frac{1}{2t}})$, $O(\sqrt{n}\log(n))$ storage for S, k, ind, separately. Summing up together, client needs storage $\widetilde{\mathcal{O}}(\lambda\sqrt{n}t^2 n^{\frac{1}{2t}})$. If we choose $t \in [2, \log_2(n)]$, client-side storage is $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$.

During the offline phase, Server 0 computes $F.\mathsf{Set}()$ function T times, so its computation is bounded by $\widetilde{\mathcal{O}}(\lambda n)$. Client computes $F.\mathsf{Gen}()$ function T times, so its computation is bounded by $\widetilde{\mathcal{O}}(\lambda\sqrt{n}tn^{\frac{1}{2n}})$. If we choose $t \in [2, \log_2(n)]$, client's computation is $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$. For the communication, Server 0 and Client communicate $\vec{h}$ and $\vec{k}$, the size of which are $\widetilde{\mathcal{O}}(\lambda\sqrt{n}t^2 n^{\frac{1}{2t}})$. If we choose $t \in [2, \log_2(n)]$, the communication overhead is $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$.

During the online phase, each Server i (for $i \in [2t]$) computes $F.\mathsf{DotProdEval}$ per query in $\widetilde{\mathcal{O}}(\sqrt{n})$. Client computes $F.\mathsf{Punc}$ twice per query in $\widetilde{\mathcal{O}}(\sqrt{n}+t^2 n^{1/2t})$. If we choose $t \in [2, \log_2(n)/2]$, the client-side computation is $\widetilde{\mathcal{O}}(\sqrt{n})$ per query. The communication between servers and Client is bounded by $\widetilde{\mathcal{O}}(\sqrt{n}t)$. Since Client receives $\vec{v}_i$ from each server $i \in [2t]$ and the size of $\vec{v}_i$ is bounded by $\sqrt{n}$. If we choose $t \in [2, \log_2(n)/2]$, the communication overhead is $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$. □

The correctness proof of our PIR scheme is pretty straightforward and it follows the same blueprint as other PIR correctness proofs in CGK paradigm [CGK20, LP23]. At a high level, we prove the client always maintains a state containing T random PMPRS keys. In each online phase the client finds a key k containing its query x with probability $1 - \mathsf{negl}.(\lambda)$. Using the key k and its hint bit the client retrieves the correct database bit $DB[x]$ and it replenishes the used key and hint bit, where the correctness of our PMPRS scheme ensures the correctness of the online phase of our construction.

Extending our PIR scheme for arbitrary n. The proposed $2t$ server PIR scheme with client preprocessing assumes a (t, n)-PMPRS as building block. However, our PMPRS scheme gives us a construction only for parameters t, n such that $n^{1/2t}$ is an integer. To get a PIR scheme for arbitrary $n \in \mathbb{N}$ and $2t$ servers, we can find the smallest integer m greater than or equal to n such that $m^{1/2t}$ is an integer, then we have $m = O(tn)$. Now we can pad the database of size n with $m-n$ dummy elements and then use our PIR scheme based on (t, m)-PMPRS to query the modified database. For $t \in [2, \log(n)/2]$ the asymptotic complexity of offline phase and online phase of this modified protocol would remain unchanged up to polylogarithmic factors in n.

4.1 Improving PIR Communication Complexity

In our proposed PIR with client preprocessing scheme, the online phase communication is dominated by the cost of server responses - which include vectors $\vec{v}_i$ output by the $\mathsf{DotProdEval}$ algorithm. However, note that the Client is interested in learning just the $\mathsf{idx}_0, \mathsf{idx}_1, \ldots, \mathsf{idx}_{t-1}^{th}$ bits of the vectors $\vec{v}_0, \vec{v}_1, \ldots, \ldots, \vec{v}_{t-1}$ respectively, since these specific bits allow the client to compute the database xor bit of the punctured set. In our constructions these vectors $\vec{v}_0, \vec{v}_1, \ldots, \ldots, \vec{v}_{t-1}$ are of length $d, d^2, \ldots, d^t = \sqrt{n}$ respectively (where $d = n^{1/2t}$ is an integer). Hence, a natural approach to reduce communication would be to use a PIR scheme where the database on the server side are the vectors $\vec{v}_0, \vec{v}_1, \ldots, \ldots, \vec{v}_{t-1}$ with client query indexes $\mathsf{idx}_0, \mathsf{idx}_1, \ldots, \mathsf{idx}_{t-1}^{th}$, instead of downloading the entire

vectors on to the client. However, there exist no non-trivial information-theoretic PIR schemes in the single server model [BIKM99].

Hence, our next approach would be to consider $4t$ servers instead of $2t$ servers - two servers for each server in the original PIR scheme. For every server and its copy the client sends the same online query - and hence these pair of servers compute the same vector $\vec{v}_i$ as output of $\mathsf{DotProdEval}$ in the online phase. And now the Client can use a 2-server PIR scheme to retrieve just the bit of interest $\vec{v}_i[\mathsf{idx}_i]$ in sublinear communication and linear computation in the database size. Instantiating the 2-server PIR primitive with the most communication efficient information-theoretic PIR due to Dvir and Gopi [DG16] gives us the following result:

Theorem 13. *There exists a $4t$-server PIR with client preprocessing protocol with threshold 1 with $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$ client storage; $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$ online client complexity; $2^{0.5(\log(n)+O(\sqrt{\log n}))}$ online server complexity and $n^{1/2d+o(1)}$ online bandwidth per query.*

Acknowledgements We thank all anonymous reviewers for their valuable comments. This work was done while all authors Jaspal Singh, Yu Wei and Vassilis Zikas were at Purdue University. Yu Wei and Vassilis Zikas were funded in part by NSF grant No. 2055599, AnalytiXIN, and Sunday Group, Inc. Jaspal Singh was funded in part by AnalytiXIN, and Sunday Group, Inc.

References

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society Press, May 2018.

[AS16] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569, 2016.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Heidelberg, April 2015.

[BI01] Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In *Automata, Languages and Programming: 28th International Colloquium, ICALP 2001 Crete, Greece, July 8–12, 2001 Proceedings 28*, pages 912–926. Springer, 2001.

[BIKM99] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. One-way functions are essential for single-server private information retrieval. In *31st ACM STOC*, pages 89–98. ACM Press, May 1999.

[BIKR02] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *43rd FOCS*, pages 261–270. IEEE Computer Society Press, November 2002.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 55–73. Springer, Heidelberg, August 2000.

[CGHK22] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2022.

[CGK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 44–75. Springer, 2020.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 402–414. Springer, Heidelberg, May 1999.

[DG16] Zeev Dvir and Sivakanth Gopi. 2-server pir with subpolynomial communication. *Journal of the ACM (JACM)*, 63(4):1–15, 2016.

[DGI+19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. Pir-psi: scaling private contact discovery. *Cryptology ePrint Archive*, 2018.

[Efr09] Klim Efremenko. 3-query locally decodable codes of subexponential length. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 39–44. ACM Press, May / June 2009.

[FLLP24] Ben Fisch, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. Single server pir via homomorphic thorp shuffles. *Cryptology ePrint Archive*, 2024.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Heidelberg, May 2014.

[GZS24] Ashrujit Ghoshal, Mingxun Zhou, and Elaine Shi. Efficient pre-processing pir without public-key cryptography. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 210–240. Springer, 2024.

[HHCG+23] Alexandra Henzinger, Matthew M Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast {Single-Server} private information retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3889–3905, 2023.

[HHG13] Ryan Henry, Yizhou Huang, and Ian Goldberg. One (block) size fits all: Pir and spir with variable-length records via multi-block queries. In *NDSS*, 2013.

[HSW23] Laura Hetz, Thomas Schneider, and Christian Weinert. Scaling mobile private contact discovery to billions of users. *Cryptology ePrint Archive*, 2023.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In László Babai, editor, *36th ACM STOC*, pages 262–271. ACM Press, June 2004.

[ISW24] Yuval Ishai, Elaine Shi, and Daniel Wichs. Pir with client-side preprocessing: Information-theoretic constructions and lower bounds. In *Annual International Cryptology Conference*, pages 148–182. Springer, 2024.

[KCG21] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 875–892, 2021.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings 38th annual symposium on foundations of computer science*, pages 364–373. IEEE, 1997.

[LG15] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 168–186. Springer, Heidelberg, January 2015.

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

[MOT+11] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. {PIR-Tor}: Scalable anonymous communication using private information retrieval. In *20th USENIX security symposium (USENIX security 11)*, 2011.

[MSR23] Muhammad Haris Mughees, I Sun, and Ling Ren. Simple and practical amortized sublinear private information retrieval. *Cryptology ePrint Archive*, 2023.

[MW22] Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy*, pages 930–947. IEEE Computer Society Press, May 2022.

[SACM21] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce M. Maggs. Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 641–669, Virtual Event, August 2021. Springer, Heidelberg.

[SWP09] DR Stinson, Ruizhong Wei, and Maura B Paterson. Combinatorial batch codes. *Advances in Mathematics of Communications*, 3(1):13–27, 2009.

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *Journal of the ACM (JACM)*, 55(1):1–16, 2008.

[ZPSZ23] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server pir with sublinear server computation. *Cryptology ePrint Archive*, 2023.