Puncturable Pseudorandom Sets and Private Information Retrieval with Poly-logarithmic Bandwith and Sublinear Time

Abstract

Imagine that one or more non-colluding servers each holds a large public database, e.g., the repository of DNS entries. Clients would like to access entries in this database without disclosing their queries to the servers. Classical private information retrieval (PIR) schemes achieve polylogarithmic bandwidth per query, but require the server to perform linear computation per query, which is a deal-breaker with respect to practical adoption.

Several recent works have shown, however, that by introducing a one-time, per-client, offline preprocessing phase, an *unbounded* number of client queries can be subsequently served with sublinear on-line computation time per query (and the cost of the preprocessing can be amortized over the unboundedly many queries). Unfortunately, existing preprocessing PIRs make undesirable tradeoffs to achieve sublinear online computation: they either require $\sqrt{n}$ or more bandwidth per query, which is asymptotically worse than classical PIR schemes, or they require the servers to store a linear amount state per client (or even per query), or require polylogarithmically many non-colluding servers.

We propose a novel 2-server preprocessing PIR scheme that achieves $\widetilde{O}(\sqrt{n})$ online computation per query, while preserving the polylogarithmic online bandwidth of classical PIR schemes. In our construction, each server stores only the original database and nothing extra, and each online query is served within a single round trip. Our construction relies on the standard LWE assumption. As an important stepping stone, we propose new, more generalized definitions for a cryptographic object called a Privately Puncturable Pseudorandom Set, and give novel constructions that depart significantly from prior approaches.

1 Introduction

Imagine that a service provider has a large, public database DB, and it is serving clients who would come and fetch a certain record from the database. For example, for a Google-like scenario, each entry of the database DB may be the search result for a specific keyword, and the client wants to look up the database by keyword; in a DNS-type scenario, each database entry contains the IP address for a specific domain name. Without loss of generality, henceforth we assume that the database $\mathsf{DB} := \{0,1\}^n$ is represented as a string of bits indexed by $\{0, 1, \ldots, n-1\}$, and a client's query is represented by an index $i \in \{0, 1, \ldots, n-1\}$ into DB (note that if the query is a keyword or domain name, it can always be hashed to an index).

Although the database itself is public, the clients wish to hide their queries from the server. This problem has been studied in a beautiful line of work called Private Information Retrieval (PIR), first formulated by Chor, Goldreich, Kushilevitz, and Sudan [CGKS95, CKGS98]. Since then, a rich line of work [CG97a, Cha04, GR05, CMS99, CG97b, KO97, Lip10, OS07, Gas04, DG16, PR93, DCIO98, BLW17, BGI16, PPY18, IKOS04, HH17, ACLS18, IKOS06, LG15, DHS14] has improved the original construction by Chor et al. [CGKS95]. Henceforth in the paper, we focus on *2-server PIR*[1], i.e.,

[1] From the long line of work on PIR [CG97a, Cha04, GR05, CMS99, CG97b, KO97, Lip10, OS07, Gas04, DG16, PR93,

there are two non-coluding servers, and we want that each individual server cannot learn anything about the clients' queries from the queries it receives.

It is well-known that we can construct (single- or multi-server) PIR schemes with *polylogarithmic* bandwidth overhead and *linear server work* per query [CG97a, Cha04, GR05, CMS99, CG97b, KO97, Lip10, OS07, PR93, BLW17, BGI16, PPY18, DG16, IKOS04, HH17]. While these PIR schemes achieve non-trivial and elegant asymptotic bounds, the prohibitive server running time per query is a deal-breaker in the path towards practical deployment. For example, in our motivating applications, the database can be terabytes or petabytes in size. Unfortunately, in the original formulation phrased by Chor et al. [CGKS95], the linear server work is inevitable for achieving privacy [BIM00] — intuitively, if there is a location that the server need not read, the query is definitely not looking for that location. To avoid this drawback, a very promising direction has been suggested by a few recent works [BIM00, CK20], namely, PIR with *preprocessing*. In PIR with preprocessing, we allow each client and the server(s) to perform one-time offline preprocessing. After the preprocessing phase, we wish to support an *unbounded* number of queries from the client. The cost of the offline preprocessing can thus be amortized "away" over sufficiently many queries, and we can hope for *sublinear amortized (i.e., online) running time* per query.

Preprocessing PIR was considered in several prior works [Lip10, BIM00, PR93]. Beimel, Ishai, and Malkin [BIM00] were the first to suggest to use preprocessing to reduce the server's online computation. Beimel et al. [BIM00] constructed a statistically secure 2-server PIR scheme with n^ϵ online bandwidth and running time for some constant $\epsilon \in (0, 1)$, by having the servers preprocess the n-bit DB into an encoded version of $\mathsf{poly}(n)$ bits. The recent work by Corrigan-Gibbs and Kogan [CK20] showed that assuming the existence of one-way functions, there exists a 2-server preprocessing PIR scheme with $O(\sqrt{n}) \cdot \mathsf{poly}(\lambda)$ online bandwidth and running time — in their scheme, the servers store only the original database DB and nothing extra; but each client needs to store a "hint" of size $\sqrt{n} \cdot \mathsf{poly}(\lambda)$. Therefore, known preprocessing 2-server PIR constructions achieve sublinear online running time at the cost of increasing the online bandwidth from poly-logarithmic to roughly $\sqrt{n}$, in comparison with classical PIR schemes without preprocessing [CG97a, GR05, CMS99, CG97b, KO97, Lip10, OS07, Gas04, BGI16, DG16].

Given the state of the affairs regarding preprocessing PIR, we ask the following natural question:

> *Can we have a 2-server preprocessing PIR scheme with sublinear online running time, but matching the poly-logarithmic bandwidth of classical PIR schemes without preprocessing?*

Before we present our results and contributions, we point out that we do not consider schemes that require the server to store per-client state. For example, one strawman candidate is to use an Oblivious RAM (ORAM) scheme [GO96, Gol87, SCSL11]. During the offline phase, the client downloads the database from the server and uses a secret key to compile the database into an ORAM which is then stored on the server. This would allow queries to be supported in (poly)logarithmic running time and bandwidth per online query, and constant roundtrips with server-side computation [DvDF+16, GHL+14, LO13, GMP16]. Unfortunately, $O(n)$ per-client state on the server would clearly be a deal-breaker in our motivating applications. Similarly, the recent doubly-efficient (1-server) PIR constructions in the designated-client setting [CHR17, BIPW17] also suffers from the same drawback, although they remove the need for the clients to store persistent state. A doubly-efficient PIR construction in the public-client setting promises to remove the $O(n)$ per-client state at the server; unfortunately, the only known construction relies on virtual blackbox (VBB) obfuscation which is known to be impossible [BGI+01]. We compare with additional related works in Section 3.

DCIO98, BLW17, BGI16, PPY18, IKOS04, HH17, ACLS18, IKOS06, LG15, DHS14], it appears that the 2-server setting is more promising in terms of eventually leading to a practical construction.

Our results and contributions. We answer the above question affirmatively, assuming the Learning With Errors (LWE) assumption. At a very high level, our scheme works as follows — henceforth we refer to the two servers as the left and right server, respectively.

- During the offline preprocessing, the client sends a single message of size roughly $\widetilde{O}(\sqrt{n})$ to the left server, where we use $\widetilde{O}(\cdot)$ to hide poly-logarithmic terms and terms related to the strength of the underlying cryptographic primitive. The left server responds with a *hint* of $\widetilde{O}(\sqrt{n})$ bits, which is stored by the client. At this moment, the online queries begin.
- For each online query, the client sends a single, poly-logarithmically sized message to each server in parallel. Specifically, the message sent to the right server is used for answering the query, that is, using its local hint and the right server's response, the client can reconstruct the correct answer to the query except with negligible probability. The message sent to the left server is used for partially "refreshing" the client's hint. The client uses the answer from the left server and the outcome of the present query to upate one entry in the $\widetilde{O}(\sqrt{n})$-sized hint it stores.

More formally, we prove the following theorem:

Theorem 1.1 (2-server preprocessing PIR)**.** *Assume the Learning With Errors (LWE) assumption. Then, there exists a 2-server preprocessing PIR scheme that satisfies the following performance bounds where $\chi(\lambda)$ denotes a security parameter related to the strength of the underlying cryptographic primitive*[2]*:*

- *the offline server running time is $\chi(\lambda) \cdot O(n) \cdot \mathsf{poly}\log(n, \lambda)$; the offline client running time and bandwidth is $\chi(\lambda) \cdot O(\sqrt{n}) \cdot \mathsf{poly}\log(n, \lambda)$;*
- *the online server and client running time per query is $\chi(\lambda) \cdot O(\sqrt{n}) \cdot \mathsf{poly}\log(n, \lambda)$; and the online communication per query is $\chi(\lambda) \cdot \mathsf{poly}\log(n, \lambda)$;*
- *each online query can be accomplished in a single roundtrip, that is, the client sends a single message to each server in parallel, and reconstructs the answer from the two servers' responses respectively; and*
- *each server needs to store only the original database DB and no extra information; and the client needs to store $\chi(\lambda) \cdot O(\sqrt{n}) \cdot \mathsf{poly}\log(n, \lambda)$ state.*

Technical highlight. Our 2-server preprocessing PIR scheme is inspired by the very recent work of Corrigan-Gibbs and Kogan [CK20]. At a very high level, their work shows how to construct a 2-server preprocessing PIR scheme using a cryptographic object which they call a Puncturable Pseudorandom Set (PRSet). At a high level, in a PRSet scheme, one can generate a secret key sk that can be used to generate a pseudorandom subset $\mathbf{Set}(\mathsf{sk}) \subseteq \{0, 1, \ldots, n-1\}$; and thus sk serves as a succinct representation of the set $\mathbf{Set}(\mathsf{sk})$. Further, there is an efficient puncturing algorithm: suppose some element $x \in \mathbf{Set}(\mathsf{sk})$, then $\mathbf{Puncture}(\mathsf{sk}, x)$ outputs a punctured key sk_x that effectively removes x from the set, i.e., $\mathbf{Set}(\mathsf{sk}_x) = \mathbf{Set}(\mathsf{sk}) \backslash \{x\}$.

Unfortunately, Corrigan-Gibbs and Kogan [CK20] fail to construct a PRSet scheme that is efficient in all dimensions, namely, efficient set enumeration, efficient membership test, and succinct punctured key. As a tradeoff, they opt for efficient set enumeration and efficient membership test — this allows their PIR scheme to achieve roughly $\sqrt{n}$ online running time. However, their

[2]Specifically, $\chi(\lambda)$ is polynomially bounded in λ assuming polynomial hardness of LWE, and is polylogarithmic in λ assuming sub-exponential security of LWE.

PRSet scheme adopts a trivial puncturing algorithm where the punctured key is simply the entire punctured set itself — this causes their online communication to be roughly $\sqrt{n}$, which is asymptotically worse than classical PIR schemes without preprocessing [CG97a, GR05, CMS99, CG97b, KO97, Lip10, OS07, Gas04, BGI16, DG16][3].

To achieve our stated result, an important intermediate stepping stone is to construct a new Privately Puncturable Pseudorandom Set (PRSet) scheme that is efficient in all dimensions. Unfortunately, as we will explain later in Section 2, these requirements seem to be inherently conflicting, and we were not able to directly reconcile them — likely Corrigan-Gibbs and Kogan [CK20] encountered the same barriers too.

Our key insight is to observe that Corrigan-Gibbs and Kogan's formulation of a PRSet scheme seems too restrictive. We generalize their PRSet abstraction in the following ways.

1. *Emulating a customized sampling distribution.* First, Corrigan-Gibbs and Kogan consider only PRSet schemes that emulate simple distributions: for example, sample a random $\sqrt{n}$-sized subset among n elements, or sample each element at random with probability $1/\sqrt{n}$.

   By contrast, we generalize the PRSet definition to allow it to emulate an arbitrary distribution of choice — and later on we shall discuss challenges of choosing this distribution.

2. *Relaxed correctness definition.* Second, Corrigan-Gibbs and Kogan's definition insists on almost-always correctness. We observe that a weaker notion which we call "occasional correctness" is sufficient for obtaining a 2-server preprocessing PIR scheme (since our PIR construction relies on parallel repetition to amplify the correctness to $1-\mathsf{negl}(\lambda)$). Specifically, we want the puncturing algorithm to remove the point x being punctured, and only the point x — but we only need this good event to happen with considerable but not overwhelming probability.

Therefore, one technical contribution we make is to figure out a more generalized/relaxed abstraction of a Privately Puncturable Pseudorandom Set (PRSet) scheme that is suitable and sufficient for constructing an efficient 2-server preprocessing PIR. Not only so, we also need to identify an appropriate sampling distribution that the PRSet should emulate. In our carefully chosen distribution, each element from $\{0, 1, \ldots, n-1\}$ is included in the set with roughly $1/(\sqrt{n} \cdot \mathsf{poly}\log n)$ probability, but the sampling is not completely independent among the elements. For example, if some element x is included in the set, it might make some other element y more or less likely to be included. As we explain in more detail in Section 2, an independent distribution seems to facilitate efficient membership test but precludes efficient set enumeration; on the other hand, having more dependence and the right type of dependence among elements can enable efficient set enumeration, but may destroy the efficient membership test. We seek middle ground by choosing a distribution that has a limited amount of dependence, and the right type of dependence among the elements.

We next show how to construct a novel PRSet scheme that emulates our carefully chosen distribution, and prove the construction secure under our new definitions. Our construction relies on the existence of Privately Puncturable PRFs which can be constructed from the LWE assumption [BLW17, BKM17, CC17, BTVW17]. Intriguingly, our PRSet construction is remotely inspired by the line of work on designing block ciphers and format preserving encryption from pseudorandom functions [RY13, MR14, SS12]; but of course, our problem definition and solutions are novel, and different in nature from this prior line of work [RY13, MR14, SS12].

Finally, we construct a 2-server preprocessing PIR scheme from our PRSet scheme and prove it correct and secure. Our construction is inspired by Corrigan-Gibbs and Kogan [CK20] but

[3] Corrigan-Gibbs and Kogan [CK20] also show that if the scheme only needs to support a single online query, then the online communication can be further reduced to polylogarithmic. Unfortunately, in this case, the linear offline running time needs to be charged to this single query under the notion of amortized cost.

differs in several important details. The proofs are rather technical and involved; and perhaps somewhat surprisingly, proving correctness turns out to be the most technical part of our proof (although proving privacy is also non-trivial). Specifically, our PIR scheme runs k parallel instances of a single-copy PIR scheme. We need to prove occasional correctness of each single-copy scheme, and use majority voting among all instances to amplify correctness. Unfortunately, we cannot easily argue occasional correctness of the single-copy PIR from the occasional correctness of the PRSet scheme. Part of challenge arises from the fact that conditioning on events that have taken place would skew the distribution of the pseudorandom sets, and we need to make an occasional correctness argument even for this skewed distribution (which does not even have a clean and succinct description). At a very high level, to make the argument work, we make a rather involved stochastic domination argument which effectively shows that conditioning on the events that have taken place will not worsen the probability of certain bad events that could lead to incorrectness. We refer the reader to Section 2.4 for more detailed discussions on the technicalities in our proof.

A practicality reality check. We do not claim the immediate practicality of our scheme. However, this work did in fact originate from our (still ongoing) quest to identify a good practical PIR scheme for a private DNS application. We believe that our work makes an important step forward towards eventually having a practical PIR scheme. Specifically, we suggest the following exciting future directions that can lend to practicality: 1) the parameters in our current theorems are not tight, therefore, concrete security parametrization is an obvious future direction; and 2) a major step towards practicality is to design a concretely efficient Privately Puncturable PRF scheme. For example, instantiations based on other assumptions can potentially be more efficient than the current LWE-based schemes.

2 Technical Overview

We now present an information technical roadmap of our construction.

2.1 Strawman Attempts

To understand our ideas, it helps to first illustrate a strawman scheme and see why it fails — the strawman scheme below is a variant (and slight simplification) of the elegant 2-server preprocessing PIR scheme by Corrigan-Gibbs and Kogan [CK20].

We shall first describe the "core" of the scheme without worrying about how to compress the storage and communication. We describe an inefficient toy scheme where we use Client, Left, and Right to denote the client, the left and right servers, respectively.

An Inefficient Toy Scheme: Single-Copy Version

Offline preprocessing.

- Client generates $\sqrt{n}$ sets $S_1, S_2, \ldots, S_{\sqrt{n}}$. $S_j \subseteq \{0, 1, \ldots, n-1\}$ where $j \in [\sqrt{n}]$ is sampled by including each element $i \in \{0, 1, \ldots, n-1\}$ with independent probability $1/\sqrt{n}$.
- Client sends the resulting sets $S_1, \ldots, S_{\sqrt{n}}$ to Left. For each set $j \in [\sqrt{n}]$, Left responds with the parity bit $p_j := \oplus_{k \in S_j} \mathsf{DB}_k$ of indices in the set.
- Client stores the hint $T := \{T_j := (S_j, p_j)\}_{j \in [\sqrt{n}]}$.

Online query for index $x \in \{0, 1, \ldots, n-1\}$.

- **Query:** (Client $\Leftrightarrow$ Left)
  1. Find an entry $T_j := (S_j, p_j)$ in its hint table T such that $x \in S_j$. Let $S^* := S_j$ if found, else let S^* be a fresh random set containing x.
  2. Send the set $S := \mathsf{Resample}(S^*, x)$ to Right, where $\mathsf{Resample}(S^*, x)$ outputs a set almost identical to S^*, except that the coin used to determine x's membership is re-tossed.
  3. Upon obtaining a response $p := \oplus_{k \in S} \mathsf{DB}_k$ from Right, output the candidate answer $\beta' := p_j \oplus p$ or $\beta' := 0$ if no such T_j was found earlier.
  4. Client obtains the true answer $\beta := \mathsf{DB}_x$ from a trusted oracle — in the full scheme, we will repeat this single-copy scheme k times, and β is computed as a majority vote among the k candidate answers, which is guaranteed to be correct except with negligible probability.
- **Refresh** (Client $\Leftrightarrow$ Left)
  1. Client samples a random set S containing x, and then lets $S' := \mathsf{Resample}(S, x)$, and sends S' to Left (notice that this is equivalent to just sampling a fresh set, but we write it this way for later convenience).
  2. Left responds with $p := \oplus_{k \in S'} \mathsf{DB}_k$. If a table entry T_j containing x was found and consumed earlier, Client replaces T_j with $(S, p \oplus \beta)$.

Correctness amplification through parallel repetition. The toy scheme guarantees correctness for the query x, provided that 1) an entry $T_j := (S_j, p_j)$ containing x is found, and 2) $\mathsf{Resample}(S_j, x)$ happens to remove x from the set S_j. It is not hard to prove that correctness is guaranted with probability at least $2/3$.

To amplify correctness, we can simply run k copies of the scheme, and instead of calling the true-answer oracle to obtain the answer β, we will set β to be the majority vote among the k candidate answers, which is correct with $1 - 2^{-\Theta(k)}$ probability. If we set $k = \omega(\log \lambda)$, then the failure probability would be negligibly small in λ.

Privacy. In the inefficient toy scheme, left-server privacy is easy to see: basically the left server Left sees $\sqrt{n}$ random sets during the offline phase. During each online query, it sees a random set as well.

Arguing right-server privacy is a little more subtle. The right server Right is not involved during the offline phase. We want to show that for each online query, Right sees a fresh random set. Recall that during a query for x, the client finds an entry $T_j := (S_j, p_j)$ such that $S_j \ni x$. It lets $S^* := S_j$ if such an entry T_j is found, else S^* is a fresh random set containing x. The client now sends $\mathsf{Resample}(S^*, x)$ to Right and if such a T_j was found and consumed, it replaces T_j with a fresh set containing x. We can prove right-server privacy by induction: suppose that conditioned on Right's view so far, the client's hint table T contains $\sqrt{n}$ independent random sets (note that this is true at the beginning of the online phase). Then, we can argue that during the next query for x, $\mathsf{Resample}(S^*, x)$ is distributed as a fresh random set conditioned on Right's view so far; and moreover, at the end of the query, the client's hint table T is distributed as $\sqrt{n}$ independent random sets conditioned on Right's view so far.

Performance bounds. In the toy scheme, the server runtime and communication is upper bounded by $O(\sqrt{n})$ for each online query. If the client adopts an efficient data structure for testing set membership, the client's runtime can also be upper bounded by $O(\sqrt{n})$ per query, but its storage is $O(n)$. We would like to reduce the online communication to polylogarithmic and reduce the client-side storage to sublinear.

Strawman ideas for improving efficiency. A failed attempt to improve efficiency is the following. Let us generate each set using a pseudorandom function (PRF) rather than using true randomness. Specifically, we may assume that the $\mathsf{PRF}(\mathsf{sk}, \cdot)$ outputs a number in in $[n]$, and an element $i \in \{0, 1, \ldots, n-1\}$ is considered in the set iff $\mathsf{PRF}(\mathsf{sk}, i) \in [1, \sqrt{n}]$. Morever, sampling a pseudorandom set would boil down to sampling a fresh PRF secret key.

In this way, a pseudorandom set can be succinctly represented by a PRF secret key, and we can improve the client's storage to $\sqrt{n} \cdot \chi(\lambda)$ where $\chi(\lambda)$ is an upper bound on the length of the PRF key. During the online phase, the client needs to resample the set at the point x where $x \in \{0, 1, \ldots, n-1\}$ is the current query. If we could represent this locally resampled set succinctly too, then we can reduce the online bandwidth.

To achieve this, our idea is to adopt a Privately Puncturable PRF [BLW17, BKM17, CC17]. A Puncturable PRF is a PRF with the following additional functionality: given a point x and the secret key sk, one can call the $\mathsf{sk}_x \leftarrow \textbf{Puncture}(\mathsf{sk}, x)$ function to obtain a secret key sk_x that allows one to evaluate the PRF correctly at any point other than x. In an ordinary Puncturable PRF construction [GGM86], using the punctured key sk_x to evaluate over the point x could result in an invalid symbol $\perp$. In contrast, a *Privately* Puncturable PRF allows one to remove a point x and obtain a punctured key sk_x; however, the punctured key sk_x does not disclose the point x. For sk_x to hide x, it must be that using sk_x to evaluate over the point x yields a non-$\perp$ outcome r. Not only so, imprecisely speaking, to a computationally bounded adversary, calling $\textbf{Puncture}(\mathsf{sk}, x)$ should behave just like resampling the PRF's outcome at the point x.

If we use a Privately Puncturable PRF to construct a pseudorandom set like above, during each online query, we obtain a construct which we call a *Privately Puncturable Pseudorandom Set*. Generating a pseudorandom set is achieved by sampling a PRF key sk. Further, given a set represented by sk that contains a specific element x, one can perform a puncturing operation at x to derive a punctured secret key sk_x — this puncturing procedure acts as if we resampled the coins that determine whether x is in the set or not.

With such a Privately Puncturable Pseudorandom set, during each online query, the client can find a secret key sk from its table T that contains the queried element $x \in \{0, 1, \ldots, n-1\}$ (or sample a random sk containing x if not found), puncture the element x from the set sk, and send the punctured key sk_x to the right server. Similarly, to perform a refresh operation with the left server, the client simply samples a key sk' such that the associated set contains x, puncture x from sk', and send the resulting punctured key sk'_x to the left server. This approach allows us to compress the online communication to $O(\chi(\lambda))$ bits per copy (and recall that there are $k = \omega(\log \lambda)$ parallel copies), where $\chi(\lambda)$ denotes the length of a punctured key.

Unfortunately, this strawman idea completely fails because to generate the set from a secret key sk, the server would have to do a linear amount of work! This defeats our original goal of achieving sublinear online runtime.

Corrigan-Gibbs and Kogan's variant and why it fails too. At this point, we also briefly overview the approach of Corrigan-Gibbs and Kogan [CK20]. They adopt a different PRSet construction that indeed allows efficient set enumeration in roughly $\sqrt{n}$ (rather than linear in n) time.

Unfortunately, their scheme does not offer a puncturing procedure that achieves any non-trivial efficiency; thus in each online query, the client has to send an entire $(\sqrt{n}-1)$-sized set (rather than a punctured secret key) to each server. More specifically, Corrigan-Gibbs and Kogan [CK20] use a Pseudorandom Permutation (PRP) on the domain $\{0,1,\ldots,n-1\}$ to sample a pseudorandom set. A secret key sk of a PRP scheme defines a corresponding set $\{\mathsf{PRP}(\mathsf{sk},i)\}_{i\in\{0,1,\ldots,n-1\}}$. Thus, the definition of the set itself gives an efficient set enumeration algorithm. To determine whether an element $x \in \{0,1,\ldots,n-1\}$ is in the set generated by sk, simply check whether $\mathsf{PRP}^{-1}(\mathsf{sk},x) \in \{0,1,\ldots,n-1\}$. Their approach samples the set from a different distribution from our earlier strawman — in particular, the sampled set is of fixed size $\sqrt{n}$, and therefore x being in the set is not independent of whether $y \neq x$ is in the set (even when the PRP is replaced with a completely random permutation). For this reason, during the online phase, the adopt a slightly different approach from our earlier strawman: after finding a set either from the table T or freshly generated that contains the queried element x, they remove x from the set with high probability, but with a small probability, they remove a random other element instead of x. In this way, the right server sees a random set of size exactly $\sqrt{n}-1$, and the same applies to the left server.

The main problem with their approach is that it is not amenable to puncturing (with non-trivial efficiency). In fact, Boneh, Kim, and Wu proved the non-existence of Puncturable PRPs [BKW17]. In our case, the domain size n is polynomially bounded, and even if we punctured a point x from the PRP, the adversary can easily recover $\mathsf{PRP}(\mathsf{sk},x)$ by evaluating $\mathsf{PRP}(\mathsf{sk},\cdot)$ at all other points.

To get around the non-existence of puncturable PRP barrier, one might be tempted to compute the pseudorandom set as $\{\mathsf{PRF}(\mathsf{sk},i)\}_{i\in\{0,1,\ldots,n-1\}}$ instead, i.e., essentially the "dual" of our earlier PRF-based strawman scheme. While this approach allows for efficient set enumeration, it precludes an efficient membership test algorithm which, in our context, would make the client's online runtime linear.

2.2 Generalized Privately Puncturable Pseudorandom Set with Occasional Correctness

To summarize the above discussion, we would like to construct a Privately Puncturable Pseudorandom Set (PRSet) scheme with some non-trivial security and efficiency requirements which we shall state shortly after defining the syntax:

- $(\mathsf{sk},\mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$: given the security parameter 1^λ and the universe size n, samples a secret key sk and a corresponding master secret key msk;
- $S \leftarrow \mathbf{Set}(\mathsf{sk})$: a deterministic algorithm that outputs a set S given the secret key sk;
- $b \leftarrow \mathbf{Member}(\mathsf{sk},x)$: given a secret key sk and an element $x \in \{0,1,\ldots,n-1\}$, output a bit indicating whether $x \in \mathbf{Set}(\mathsf{sk})$; and
- $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk},x)$: given a master secret key msk, outputs a punctured secret key sk_x.

Efficiency requirements. Our goal is to use the PRSet scheme to sample pseudorandom sets of size roughly $\sqrt{n}$. For efficiency, we want that enumerating the set can be accomplished with the $\mathbf{Set}(\mathsf{sk})$ algorithm, taking time roughly $\sqrt{n}$ (rather than linear in n). Additionally, we want that the membership test algorithm, i.e., $\mathbf{Member}(\mathsf{sk},x)$, completes in polylogarithmic time.

Security requirements. For security, we want the following:

1. **Pseudorandomness**: given a randomly sampled secrete key $(\mathsf{sk}, \_) \leftarrow \mathbf{Gen}(1^\lambda, n)$, the associated set $\mathbf{Set}(\mathsf{sk})$ is computationally indistinguishable from a set sampled at random from some distribution $\mathcal{D}_n$ — we shall specify the distribution $\mathcal{D}_n$ later;
2. **Security w.r.t. puncturing I**: we want the following two distributions to be computationally indistinguishable for any $x \in \{0, 1, \ldots, n-1\}$:
   - Sample $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $\mathbf{Set}(\mathsf{sk})$ contains x, and output $\mathbf{Puncture}(\mathsf{msk}, x)$.
   - Sample $(\mathsf{sk}, \_) \leftarrow \mathbf{Gen}(1^\lambda, n)$ and output sk.

   The above definition implies that a punctured secret key should be simulatable without knowledge of the point x being punctured.
3. **Security w.r.t. puncturing II**: we want the following two distributions to be computationally indistinguishable for any $x \in \{0, 1, \ldots, n-1\}$:
   - Sample $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $\mathbf{Set}(\mathsf{sk})$ contains x, let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$, and output $(\mathbf{Set}(\mathsf{sk}), x \in \mathbf{Set}(\mathsf{sk}_x))$.
   - Sample $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $\mathbf{Set}(\mathsf{sk})$ contains x, and output $(\mathbf{Set}(\mathsf{sk}), \mathsf{Bernoulli}(\rho))$ where $\rho := \Pr_{S \xleftarrow{\$} \mathcal{D}_n}[x \in S]$.

   Intuitively, the above says that knowing the unpunctured set reveals nothing about whether x still belongs to the set after puncturing x from the set.

Remark 1. Jumping ahead, the "security w.r.t. puncturing I" property will be used in proving the privacy of our PIR scheme, and the "security w.r.t. puncturing II" property will be needed for proving correctness — it turns out that the correctness proof is rather technical (see Section 2.4 for further discussions).

From the strawman attempts described in Section 2.1, we are essentially faced with the following dilemma. Consider some distribution $\mathcal{D}_n$ which the pseudorandom set tries to emulate. One one hand, we want each element to be included in the set with *independent* probability, since this would enable puncturing and efficient membership test. On the other hand, we do not want complete independence among elements, since it would preclude efficient set enumeration. It may seem like we have hit a wall, but what comes to our rescue is the observation that our single-copy scheme need not guarantee $(1-\mathsf{negl}(\lambda))$-correctness. Since we can take majority vote among $k = \omega(\log \lambda)$ parallel copies, it suffices for each copy to have $2/3$-correctness. We therefore hope to seek middle ground between the seemingly conflicting requirements by relaxing correctness. Imprecisely speaking, we require the following type of correctness guarantee w.r.t. puncturing:

Occasional correctness w.r.t. puncturing (informal). With $1 - o(1)$ probability over the choice of a PRSet secret key that contains an element $x \in \{0, 1, \ldots, n-1\}$, puncturing at an arbitrary point x would *remove the point x from the set, and only x*. Recall that earlier, we said that puncturing at x should behave as if we resampled the choice whether x is in the set or not, independent of the unpunctured set (see "security w.r.t. puncturing II"). Thus, the relaxed correctness requirement intuitively implies that the resampling that happens at puncturing should only choose to include x in the punctured set with small (but possibly non-negligible) probability. Furthermore, jumping ahead, in our construction, puncturing at x may occasionally end up removing other elements besides x from the set, but this should not happen too often.

It turns out that to formally prove our PIR scheme secure, we actually need a more refined occasional correctness definition. Specifically, our formal definition lets us specify exactly which set

of elements are related to x such that they might accidentally get evicted from the set due to the puncturing of x. Further, we also need define an extra monotinicity condition that the puncturing operation never adds element to the set. We defer the formal definition to Section 5.1 (specifically, see the "functionality preservation under puncturing" notion).

Choosing a sampling distribution. Recall that each element wants to decide at random whether to be included in the sampled set. Our idea is to allow weak dependence in the coins chosen by different elements. Such weak dependence should be sufficient to allow efficient set enumeration, and yet without destroying the ability to efficiently test membership. Of course, we have to pay a price for introducing the weak dependence among elements, and indeed we pay in terms of the correctness of puncturing. In our PRSet scheme, puncturing a secret key msk at a point x may, with some small but non-negligible probability over the choice of msk, not only cause the coins for x to be resampled, but also the coins for some elements other than x. When this happens, puncturing at the point x may end up removing other elements from the set as well, and thus lead to an incorrect output in our single-copy PIR scheme.

Even with this high-level intutition, identifying a construction that works is challenging. To this end, our approach is very remotely inspired by the line of work on designing block ciphers and format-preserving encryption [RY13, MR14, SS12]. Despite the remote reminiscence, of course, our problem definition and solutions are fundamentally different from block ciphers.

To convey the intuition, let us first describe the distribution our PRSet scheme aims to emulate, assuming the existence of a random oracle $\mathsf{RO} : \{0,1\}^* \to \{0,1\}$. Suppose we sample an RO at random which will determine a pseudorandom set of expected size roughly $\sqrt{n}/\log^2 n$. To determine if an element $x \in \{0, 1, \ldots, n-1\}$ is in the set associated with RO or not, write x as a $\log n$-bit string, i.e., $x := \{0,1\}^{\log n}$. We then say that x is in the set iff using RO to "hash" every sufficiently long suffix of $0^{2\log\log n}||x$ outputs 1. More formally, set membership of $x \in \{0,1\}^{\log n}$ is defined with the following algorithm:

1. let $z := 0^B||x$, i.e., we prepend $B := \lceil 2\log\log n \rceil$ number of 0s in front of the string x;
2. we say that x is in the set iff $\mathsf{RO}(z[i:]) = 1$ for every $i \in [1, \frac{1}{2}\log n + B]$, where $z[i:]$ denotes the suffice $z[i : \log n + B]$ starting at the index i. For example, $z[1:] = z$, $z[2:]$ is the string z removing the first bit, and so on.

As a toy example, suppose that $n = 4$, and thus $B = 2\log\log n = 2$, and $\frac{1}{2}\log n + B = 3$. Then, the string $x = 0110$ is in the set iff $\mathsf{RO}(00110) = \mathsf{RO}(0110) = \mathsf{RO}(110) = 1$.

The above sampling distribution has the following properties:

- *Expected set size.* Each element $x \in \{0,1\}^{\log n}$ is included in the set with probability $2^{-(\frac{1}{2}\log n + B)} \approx 1/(\sqrt{n}\log^2 n)$, and thus the expected set size is roughly $\sqrt{n}/\log^2 n$.
- *Efficient membership test.* The definition itself gives an efficient algorithm to test if an element $x \in \{0,1\}^{\log n}$ is in the set, by making $\frac{1}{2}\log n + B$ calls to RO.
- *Efficient set enumeration.* Enumerating all elements in the set can be accomplished by making roughly $\sqrt{n} \cdot \mathsf{poly}\log n$ calls to RO with at least $1 - o(1)$ probability. Let $\ell \geq \frac{1}{2}\log n + 1$, and let Z_ℓ be the set of all strings z of length exactly ℓ, such that using RO to "hash" all suffixes of z of length at least $\frac{1}{2}\log n + 1$ outputs 1. To enumerate the set generated by RO, we can start out $Z_{\frac{1}{2}\log n+1}$, which takes at most $2^{\frac{1}{2}\log n+1}$ RO calls to generate. Then, for each $\ell := \frac{1}{2}\log n + 2$ to $\frac{1}{2}\log n + B$, we will generate Z_ℓ from $Z_{\ell-1}$. This can be accomplished by enumerating all elements $z' \in Z_{\ell-1}$, and checking whether $\mathsf{RO}(0||z') = 1$ and $\mathsf{RO}(1||z') = 1$. In our subsequent

formal technical sections, we will prove that with at least $1 - o(1)$ probability, all the Z_ℓ sets encountered along the way will not exceed $\sqrt{n} \cdot \mathsf{poly}' \log n$ in size. Thus, with at least $1 - o(1)$ probability, set enumeration can be accomplished by making at most $\sqrt{n} \cdot \mathsf{poly} \log n$ calls to RO.

- *Occasional correctness of "puncturing".* Suppose that we sample an RO whose associated set contains the element $x \in \{0,1\}^{\log n}$. In this idealized world with RO, imagine that puncturing the point x from RO means that we resample the outcomes for $\mathsf{RO}((0^B||x)[i:])$ for every $i \in [1, \frac{1}{2}\log n + B]$. We want to make sure that with $1 - o(1)$ probability over the choice of the RO, puncturing the point x removes x and only x from the resulting set. We prove this statement in our subsequent technical sections. At a high level, to prove this statement, it suffices to prove that the expected number of related elements in the set is $o(1)$, where an element $x' \neq x$ is related to x, iff the longest common suffix of x and x' has length at least $\frac{1}{2}n + 1$.

Our PRSet scheme. Given the above distribution $\mathcal{D}_n$, we can derive a PRSet scheme by replacing the RO with a privately puncturable PRF [BLW17, BKM17, CC17]. Puncturing a point $x \in \{0,1\}^{\log n}$ simply punctures all queries we must make to the PRF to determine x's membership. For a punctured key to be indistinguishable from a fresh generated secret key, we puncture a set of "useless" points from a freshly generated secret key as well. More formally, let $\mathsf{PRF} := (\mathbf{Gen}, \mathbf{Eval}, \mathbf{Puncture}, \mathbf{PEval})$ be a privately puncturable PRF scheme where **Eval** and **PEval** denote the evaluation algorithms using a normal key and a punctured key, respectively. Let $B := \lceil 2 \log \log n \rceil$. Our PRSet scheme works as follows:

Our PRSet scheme

- $\mathbf{Gen}(1^\lambda, n)$:
  1. call $\mathsf{PRF}.\mathbf{Gen}$ with the appropriate parameters to generate a normal PRF key sk'.
  2. let P be an arbitrary set of $\frac{1}{2}\log n + B$ distinct strings in $\{0,1\}^{\log n + B}$ that begin with the bit 1;
  3. call $\mathsf{sk} \leftarrow \mathsf{PRF}.\mathbf{Puncture}(\mathsf{sk}', P)$, and output $(\mathsf{sk}, \mathsf{msk} = \mathsf{sk}')$.
- $\mathbf{Set}(\mathsf{sk})$: similar to the earlier set enumeration algorithm for the distribution $\mathcal{D}_n$, but now replace $\mathsf{RO}(\cdot)$ calls with calls to $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}, \cdot)$ instead;
- $\mathbf{Member}(\mathsf{sk}, x)$:
  1. write $x \in \{0,1\}^{\log n}$ as a binary string, and let $z := 0^B||x$;
  2. if for every $1 \leq i \leq \frac{1}{2} \cdot \log n + B$, $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}, z[i:]) = 1$, then output 1; else output 0.
- $\mathbf{Puncture}(\mathsf{msk}, x)$:
  1. write $x \in \{0,1\}^{\log n}$ as a binary string, and let $z := 0^B||x$;
  2. let $P := \{z[i:]\}_{i \in [1, \frac{1}{2} \cdot \log n + B]}$ and $\mathsf{sk}_P \leftarrow \mathsf{PRF}.\mathbf{Puncture}(\mathsf{msk}, P)$; output sk_P.

Note that the punctured set is logarithmic in size, and this is necessary for the punctured key to be succinct. Specifically, each sk has size $\chi(\lambda) \cdot \log n$ using known Privately Puncturable PRF constructions [BKM17, CC17], where $\chi(\lambda)$ is related to the strength of the underlying cryptographic assumption as mentioned earlier. We defer to Section 5 the detailed proof of security for our PRSet scheme and the probabilistic analysis of the distribution $\mathcal{D}_n$ that it emulates.

2.3 Putting it All Together: Our PIR Scheme

Putting everything together, we describe our PIR scheme below where Client, Left, Right denote the client, the left server, and the right server, respectively.

Our PIR Scheme

Run $k = \omega(\lambda)$ parallel copies of the single-copy scheme described below.

Offline setup. For $i = 1$ to $\mathsf{lenT} := 6\sqrt{n} \cdot \log^3 n$ in parallel:

1. Client: Sample $(\mathsf{sk}_i, \mathsf{msk}_i) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$, send sk_i to Left.
2. Left: Run $S_i \leftarrow \mathsf{PRSet.Set}(\mathsf{sk}_i)$. If the runtime of $\mathsf{PRSet.Set}(\mathsf{sk}_i)$, measured in terms of $\mathsf{PRF.PEval}$ calls, exceeds $\mathsf{maxT} := 6\sqrt{n}\log^5 n$, return $p_i := 0$ to Client. Else, return the parity bit $p_i \in \{0,1\}$ of the set S_i to Client.
3. Client: Save $T_i := (\mathsf{sk}_i, \mathsf{msk}_i, p_i)$ where $T := (T_1, T_2, \ldots, T_{\mathsf{lenT}})$ denotes a table saved by Client.

Online query for index $x \in \{0, 1, \ldots, n-1\}$.

- **Query** (Client $\Leftrightarrow$ Right):
  1. Client:
     - Sample a $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$ subject to $\mathsf{PRSet.Member}(\mathsf{sk}, x) = 1$, append $(\mathsf{sk}, \mathsf{msk}, 0)$ to the end of the table T.
     - Henceforth parse $T_i := (\mathsf{sk}_i, \mathsf{msk}_i, p_i)$. Let j be the smallest entry in the table T such that $\mathsf{PRSet.Member}(\mathsf{sk}_j, x) = 1$.
     - Call $\widetilde{\mathsf{sk}}_j := \mathsf{PRSet.Puncture}(\mathsf{msk}_j, x)$. Send $\widetilde{\mathsf{sk}}_j$ to Right.
  2. Right: Run $S \leftarrow \mathsf{PRSet.Set}(\widetilde{\mathsf{sk}}_j)$. If the runtime exceeds maxT, return $p := 0$ to Client. Else, return the parity bit $p \in \{0,1\}$ of the set S to Client.
  3. Client: Let $\beta' := p \oplus p_j$ be a candidate answer of this copy. Let β be the majority vote among the candidate answers of all k copies.
- **Refresh** (Client $\Leftrightarrow$ Left):
  1. Client:
     - Sample a new $(\mathsf{sk}', \mathsf{msk}') \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$ subject to $\mathsf{PRSet.Member}(\mathsf{sk}', x) = 1$.
     - Call $\mathsf{sk}'_x \leftarrow \mathsf{PRSet.Puncture}(\mathsf{msk}', x)$, and send sk'_x to Left.
  2. Left: Run $S \leftarrow \mathsf{PRSet.Set}(\mathsf{sk}'_x)$. If the runtime exceeds maxT, return $p := 0$ to Client. Else, return the parity bit $p \in \{0,1\}$ of the set S to Client.
  3. Client: Replace $T_j := (\mathsf{sk}', \mathsf{msk}', p \oplus \beta)$. Finally, remove the last entry from the table T.

2.4 Proof Roadmap

Proving our PIR scheme secure turns out to be very much non-trivial. Somewhat surprisingly at first, the most challenging part is actually the proof of occasional correctness of the single-copy version of our PIR scheme (see Sections 5.4 and 7.3) — even though we are only asking for a relaxed correctness requirement. At a high level, the challenge arises from the fact that the distribution of

the PRSet key sk becomes *skewed*, when conditioning on the fact that the key sk is chosen during the online query phase, since it is the first entry in the client's hint table T that contains the queried element x. In one part of the occasional correctness proof, we need to argue that, imprecisely speaking, despite this skewed distribution, the selected secret key can provide a correct answer to the present query with $1 - o(1)$ probability. In a key technical step (see the proof of Lemma 7.8), we make a *stochastic domination* type of argument that roughly speaking, proves the following: conditioned on the secret key not having been consumed so far and now being consumed by the present query, it makes it less likely, in comparison with a freshly generated PRSet key, for certain bad events to happen that might lead to incorrectness. To make this argument work, we rewrite the randomized experiment into an equivalent one where the sampling of a subset of the random coins is delayed to the point when they are consumed. Note also that in our scheme, multiple bad events can lead to incorrectness of a single copy of the scheme. Therefore, in our proof, we bound the probability of each of these bad events (see Sections 7.3.1 and 7.3.2) — to do so, we often view the randomized experiment in different lights, to facilititate the analyses of different bad events.

Besides the occasional correctness proof, other major parts of the proof include

1. Proving that our PRSet scheme satisfies our security definitions (see Section 5.4). Here, we adopt standard techniques from the cryptography literature (nonetheless, the proof is non-trivial);

2. Proving the privacy of our PIR scheme (see Section 7.2). The main challenge here is proving right-server privacy. Specifically, a key technical step is to argue that even when conditioned on the right-server's view so far, the client's hint table is distributed as a fresh random sample of $\sqrt{n} \cdot \mathsf{poly}\log(n)$ secret keys from $\mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)$.

3 Additional Related Work

Beimel et al. [BIM00] proved that in the original formulation of PIR, the servers must collectively probe all n bits of the database on average to respond to a client's query. Various techniques have been suggested to overcome this key performance bottleneck, e.g., encoding the server-side database, storing per-client or even per-query server state, batching, introducing assumptions like Virtual-Blackbox obfuscation which is known to be impossible [BGI+01], or having many non-colluding servers. We review this line of work below.

As mentioned in Section 1, the work of Beimel et al. achieve sublinear online computation by encoding the database into a $n^{1+\epsilon}$ to $\mathsf{poly}(n)$-sized string. The recent (designated-client) doubly efficient PIR schemes [CHR17, BIPW17] rely on encoding the database as well as having the server store $\Omega(n)$ state per client, which is a deal breaker in our motivating applications. Boyle et al. [BIPW17] show that assuming Virtual-Blackbox Obfuscation which is known to be impossible [BGI+01] (and additional non-standard assumptions that are not yet well understood), one can indeed construct a preprocessing PIR with com n^ϵ online runtime and computation, without having to store per-client state at the server.

A related notion called *private anonymous data access* was recently introduced by Hamlin et al. [HOWW19]. PANDA is a form of preprocessing PIR which requires a *third-party trusted setup* besides the client and the servers (which is not necessary in our work); and moreover, the server storage and time grows w.r.t. the number of corrupt clients. In our motivating examples, the number of clients is essentially unbounded which makes known PANDA schemes unsuitable. Some works [BIM00, DCIO98] suggested having the server store *per-query* state to reduce the online time. Specifically, the construction suggested by Beimel et al. [BIM00] requires a linear amount of server

storage per query, and this is even worse than per-client storage in our motivating applications. Other works [PPY18] improve the online time by making the number of public-key operations sublinear, along with a still *linear* number of symmetric-key operations. Sharding has also been suggested to spread out the server work online [DHS14] but the total work across all servers is still linear.

A couple works [PR93, Lip10] were able to construct preprocessing PIR schemes whose online runtime is marginally sublinear, e.g., roughly $O(n/\log n)$; and the complexity of these protocols is much larger than the very recent work of Corrigan-Gibbs and Kogan [CK20].

An elegant line of work suggested batching queries from the same client [IKOS04, HH17, ACLS18] or among multiple clients [BIM00, IKOS06, LG15] to amortize the linear server work among the batch. Our formulation can be viewed as a generalization of batched PIR, since we do not require the requests to come in a batch, and we can nonetheless achieve small online communication and work. The work by Beimel et al. [BIM00] also showed now to get a pre-processing PIR with polylogarithmic online bandwidth and cost assuming polylogarithmically many non-colluding servers, and $\mathsf{poly}(n)$ server space. Last but not the least, Toledo et al. [TDG16] consider how to relax the security definition and achieve differential-privacy-style security, to improve the server time to sublinear.

4 Preliminaries: Privately Puncturable PRFs

A Puncturable PRF scheme [BLW17, BKM17, CC17, BTVW17] consists of the following possibly randomized algorithms:

- $\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, L, m)$: takes in a security parameter λ, the maximum message length L, and the number of points to be punctured m, and samples a secret key sk.
- $y \leftarrow \mathbf{Eval}(\mathsf{sk}, x)$: given the secret key sk and an input $x \in \{0,1\}^{\leq L} := \{\emptyset\} \cup \{0,1\} \cup \{0,1\}^2 \cup \ldots \cup \{0,1\}^L$, outputs an evaluation result $y \in \{0,1\}$.
- $\mathsf{sk}_P \leftarrow \mathbf{Puncture}(\mathsf{sk}, P)$: let $P \subseteq \{0,1\}^{\leq L}$ be a set of exactly m distinct points to puncture, output a punctured key sk_P.
- $y \leftarrow \mathbf{PEval}(\mathsf{sk}_P, x)$: given a punctured key sk_P and a point $x \in \{0,1\}^{\leq L}$, outputs an evaluation result y.

In the above, **Eval** and **PEval** are both deterministic algorithms.

Functionality preservation. We say that a Puncturable PRF scheme $\mathsf{PRF} := (\mathbf{Gen}, \mathbf{Eval}, \mathbf{Puncture}, \mathbf{PEval})$ satisfies functionality preservation, iff for any L and m that are upper bouned by a fixed polynomial in λ, for any non-uniform p.p.t. stateful adversary $\mathcal{A}$ (that is required to output a set P of size exactly m), there is a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\begin{array}{c} \mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, L, m), P \leftarrow \mathcal{A}(1^\lambda), \\ \mathsf{sk}_P \leftarrow \mathbf{Puncture}(\mathsf{sk}, P), \\ x \leftarrow \mathcal{A}^{\mathbf{Eval}(\mathsf{sk},\cdot)}(\mathsf{sk}_P) \end{array} : (x \notin P) \wedge (\mathbf{Eval}(\mathsf{sk}, x) \neq \mathbf{PEval}(\mathsf{sk}_P, x)) \right] \leq \mathsf{negl}(\lambda)$$

Pseudorandomness. We say that a Puncturable PRF scheme $\mathsf{PRF} := (\mathbf{Gen}, \mathbf{Eval}, \mathbf{Puncture}, \mathbf{PEval})$ satisfies pseudorandomness iff for any L and m that are upper bouned by a fixed polynomial in λ, for any non-uniform p.p.t. stateful, *admissible* adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that the following experiments are computationally indistinguishable:

1. $\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, L, m)$, $P \leftarrow \mathcal{A}(1^\lambda)$, $\mathsf{sk}_P \leftarrow \mathbf{Puncture}(\mathsf{sk}, P)$, $b \leftarrow \mathcal{A}^{\mathbf{Eval}(\mathsf{sk},\cdot)}(\mathsf{sk}_P, \{\mathbf{Eval}(\mathsf{sk}, x)\}_{x\in P})$, and output b.
2. $\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, L, m)$, $P \leftarrow \mathcal{A}(1^\lambda)$, $\mathsf{sk}_P \leftarrow \mathbf{Puncture}(\mathsf{sk}, P)$, sample $R_1, R_2, \ldots, R_m \xleftarrow{\$} \{0,1\}$, $b \leftarrow \mathcal{A}^{\mathbf{Eval}(\mathsf{sk},\cdot)}(\mathsf{sk}_P, R_1, R_2, \ldots, R_m)$, and output b.

We say that the adversary $\mathcal{A}$ is admissible if it never queries the $\mathbf{Eval}(\mathsf{sk}, \cdot)$ oracle on any point $x \in P$, and moreover it always outputs a set P of size exactly m.

Privacy w.r.t. puncturing. We say that a Puncturable PRF scheme $\mathsf{PRF} := (\mathbf{Gen}, \mathbf{Eval}, \mathbf{Puncture}, \mathbf{PEval})$ satisfies privacy w.r.t. puncturing, iff for any L and m that are upper bouned by a fixed polynomial in λ, for any non-uniform p.p.t. stateful, *admissible* adversary $\mathcal{A}$, there is a negligible function $\mathsf{negl}(\cdot)$ such that $\mathsf{Expt}^0(1^\lambda, L, m)$ and $\mathsf{Expt}^1(1^\lambda, L, m)$ are computationally indistinguishable where the experiment $\mathsf{Expt}^b(1^\lambda, L, m)$ is defined as below:

- $\mathsf{sk} \leftarrow \mathbf{Gen}(1^\lambda, L, m)$,
- $P_0, P_1 \leftarrow \mathcal{A}(1^\lambda)$,
- $\mathsf{sk}_{P_b} \leftarrow \mathbf{Puncture}(\mathsf{sk}, P_b)$,
- $b' \leftarrow \mathcal{A}^{\mathbf{Eval}(\mathsf{sk},\cdot)}(\mathsf{sk}_{P_b})$, and output b'.

The adversary $\mathcal{A}$ is said to be *admissible* iff it never queries $\mathbf{Eval}(\mathsf{sk}, \cdot)$ on any point $x \in P_0 \cup P_1$; and moreover, it always outputs two sets P_0 and P_1 both of size exactly m.

Theorem 4.1 (Private Puncturable PRFs [BKM17, CC17, BTVW17])**.** *Suppose that the Learning With Errors (LWE) assumption is hard. Then, there exists a privately puncturable PRF scheme that achieves $\chi(\lambda) \cdot \mathsf{poly}(L, m)$ runtime for* **Gen**, **Eval**, *and* **PEval**, *and moreover, each punctured key is of length $\chi(\lambda) \cdot \mathsf{poly}(L) \cdot m$.*

As mentioned, the term $\chi(\lambda)$ is related to the strength of the LWE assumption. If we assume standard polynomial security, $\chi(\lambda)$ is polynomially bounded in λ; if we assume subexponential security, $\chi(\lambda)$ is poly-logarithmic in λ.

5 Privately Puncturable Pseudorandom Set

5.1 Definition

Suppose S is a finite set. We use the notation 2^S to denote the power set of S, i.e., a set that contains all subsets of S.

Let $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$ be a family of distributions on $2^{\{0,1,\ldots,n-1\}}$. A Privately Puncturable Pseudorandom Set (PRSet) scheme for $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$ consists of the following possibly randomized algorithms:

- $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$: takes in a security parameter 1^λ and the parameter n, and samples a secret key sk and a master secret key msk.
- $S \leftarrow \mathbf{Set}(\mathsf{sk})$: a deterministic algorithm that outputs a subset $S \subseteq \{0, 1, \ldots, n-1\}$ when given a secret key or punctured secret key sk. Henceforth, we often use the notation $\mathbf{Set}(\mathsf{sk})$ to denote the subset generated by the secret key sk.
- $\{0,1\} \leftarrow \mathbf{Member}(\mathsf{sk}, x)$: given a secret key or punctured secret key sk and an element $x \in \{0, 1, \ldots, n-1\}$, output if $x \in \mathbf{Set}(\mathsf{sk})$.
- $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$: given the master secret key msk and an element $x \in \{0, 1, \ldots, n-1\}$, outputs a punctured secret key sk_x.

Efficiency requirements. We would like the set enumerate algorithm $\mathbf{Set}(\mathsf{sk})$ and the membership testing algorithm $\mathbf{Member}(\mathsf{sk}, x)$ to satisfy certain efficiency requirements. In our application, the set size $\mathbf{Set}(\mathsf{sk})$ is concentrated around $\widetilde{O}(\sqrt{n})$ — in this case, we want to be able to enumerate the set $\mathbf{Set}(\mathsf{sk})$ in time that is roughly the size of $\mathbf{Set}(\mathsf{sk})$, up to polylogarithmic factors, rather than linear in n.

A naïve way to compute $\mathbf{Member}(\mathsf{sk}, x)$ is to call $\mathbf{Set}(\mathsf{sk})$ to enumerate the whole set and check if x is in it; however, this would take time at least linear in the size of $\mathbf{Set}(\mathsf{sk})$. We want an efficient algorithm to test membership whose runtime is polylogarithmic in n rather than linear in the size of $\mathbf{Set}(\mathsf{sk})$.

Correctness. Correctness requires that calling $\mathbf{Member}(\mathsf{sk}, x)$ to determine the membership of x in $\mathbf{Set}(\mathsf{sk})$ always returns a correct result. More formally, for every $\lambda \in \mathbb{N}$, every $n \in \mathbb{N}$, every $x, y \in \{0, 1, \ldots, n-1\}$, with probability 1, the following must hold:

- $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n) \implies \mathbf{Member}(\mathsf{sk}, x) = 1$ iff $x \in \mathbf{Set}(\mathsf{sk})$;
- $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n), \mathsf{sk}_y \leftarrow \mathbf{Puncture}(\mathsf{msk}, y) \implies \mathbf{Member}(\mathsf{sk}_y, x) = 1$ iff $x \in \mathbf{Set}(\mathsf{sk}_y)$.

Pseudorandomness w.r.t. some distribution. We say that $\mathsf{PRSet} := (\mathbf{Gen}, \mathbf{Set}, \mathbf{Member}, \mathbf{Puncture})$ satisfies pseudorandomness w.r.t. the family of distributions $\{\mathcal{D}_n\}_{n \in \mathbb{N}}$ iff for any $n \in \mathbb{N}$, the following distributions are computationally indistinguishable:

1. $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$, output $\mathbf{Set}(\mathsf{sk})$;
2. $S \leftarrow \mathcal{D}_n$, output S.

Security w.r.t. puncturing. We say that $\mathsf{PRSet} := (\mathbf{Gen}, \mathbf{Set}, \mathbf{Member}, \mathbf{Puncture})$ satisfies security w.r.t. puncturing iff for any $n \in \mathbb{N}$, for any $x \in \{0, 1, \ldots, n-1\}$, the following hold:

I) The following two distributions are computationally indistinguishable:

- Repeat $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $x \in \mathbf{Set}(\mathsf{sk})$, let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$, and output sk_x.
- $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ and output sk.

The above implies that the punctured key sk_x hides the punctured point x.

II) The following two distributions are computationally indistinguishable:

- Repeat $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $x \in \mathbf{Set}(\mathsf{sk})$, let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$, and output $(\mathbf{Set}(\mathsf{sk}), x \in \mathbf{Set}(\mathsf{sk}_x))$.
- Repeat $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $x \in \mathbf{Set}(\mathsf{sk})$, and output $(\mathbf{Set}(\mathsf{sk}), \mathsf{Bernoulli}(\rho))$ where $\rho := \Pr_{S \xleftarrow{\$} \mathcal{D}_n}[x \in S]$.

The above implies that knowing the original unpunctured set does not give information about whether x is contained in the punctured set after puncturing x.

Observe that for the **Puncture** algorithm to satisfy security w.r.t. puncturing alone is trivial, since $\mathbf{Puncture}(\mathsf{msk}, x)$ can simply sample a fresh $\mathsf{sk}^* \leftarrow \mathbf{Gen}(1^\lambda)$ and output sk^*. However, this trivial approach would result in a $\mathbf{Set}(\mathsf{sk}^*)$ that is independent of the original $\mathbf{Set}(\mathsf{sk})$. Therefore, we would like some notion of functionality preservation for **Puncture** too. Roughly speaking, given $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$, we would like the set $\mathbf{Set}(\mathbf{Puncture}(\mathsf{msk}, x))$ to be "close" to the set $\mathbf{Set}(\mathsf{sk})$.

Functionality preservation under puncturing. To define functionality preservation, we introduce a symmetric boolean predicate $\mathsf{Related}(x, y) : \{0, 1, \ldots, n-1\}^2 \to \{0, 1\}$, that outputs whether two elements x and y are related or not. We may assume that $\mathsf{Related}(x, y) = \mathsf{Related}(y, x)$.

We say that $\mathsf{PRSet} := (\mathbf{Gen}, \mathbf{Set}, \mathbf{Member}, \mathbf{Puncture})$ satisfies functionality preservation w.r.t. the $\mathsf{Related}$ predicate, iff for any $\lambda, n \in \mathbb{N}$, with probability $1 - \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$, the following holds: let $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$, then, for any $x \in \mathbf{Set}(\mathsf{sk})$: let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$:

1. $\mathbf{Set}(\mathsf{sk}_x) \subseteq \mathbf{Set}(\mathsf{sk})$;
2. $\mathbf{Set}(\mathsf{sk}_x)$ runs in time no more than $\mathbf{Set}(\mathsf{sk})$;
3. for any $y \in \mathbf{Set}(\mathsf{sk}) \backslash \mathbf{Set}(\mathsf{sk}_x)$, it must be that $\mathsf{Related}(x, y) = 1$.

Intuitively, the above requires that puncturing results in a strict subset of the original set; and the set enumeration time can only reduce once a set has been punctured. Moreover, puncturing x can only cause elements related to x to be removed from the set. Later on, when we instantiate the distribution $S \xleftarrow{\$} \mathcal{D}_n$ (i.e., the distribution that is emulated by our PRSet scheme), we shall see that most elements in the sampled set S likely do not have other related elements in S.

5.2 Choosing an Appropriate Distribution $\mathcal{D}_n$

We will construct a PRSet scheme for an appropriate distribution $\mathcal{D}_n$, and we begin by defining this distribution. Consider a random oracle RO: upon receiving a query x, if x has been queried before, RO returns the same answer as before; else RO flips a random $\mathsf{coin} \xleftarrow{\$} \{0, 1\}$, and returns coin. We now specify the distribution $\mathcal{D}_n$ which relies on RO— without loss of generality, we shall assume that n is a power of 4, and $\log(\cdot)$ means $\log_2(\cdot)$ unless otherwise stated:

Distribution $\mathcal{D}_n$

Notations. Let the block size $B := \lceil 2 \log \log n \rceil$.

Distribution $\mathcal{D}_n$. Sample RO at random.

- Let $S := \emptyset$.
- For $x \in \{0, 1, \ldots, n-1\}$:
  - Write $x \in \{0, 1\}^{\log n}$ as a binary string, let $z := 0^B || x$;
  - If for every $1 \le i \le \frac{1}{2} \cdot \log n + B$, $\mathsf{RO}(z[i:]) = 1$ where $z[i:]$ denotes the length-$(\log n + B - i + 1)$ suffix of z, then let $S := S \cup \{x\}$.
- Output S.

Fact 5.1. *For any fixed $x \in \{0, 1, \ldots, n-1\}$, $\Pr_{S \xleftarrow{\$} \mathcal{D}_n}[x \in S] = \frac{1}{\sqrt{n} \cdot 2^B}$. Moreover, $\mathbb{E}_{S \xleftarrow{\$} \mathcal{D}_n}[|S|] \le \frac{\sqrt{n}}{\log^2 n}$.*

Proof. Every element is included in the output S with probability $\left(\frac{1}{2}\right)^{\log n/2 + B} = \frac{1}{\sqrt{n} \cdot 2^B} \le \frac{1}{\sqrt{n} \cdot \log^2 n}$. Therefore, by the linearity of expectation, $\mathbb{E}_{S \xleftarrow{\$} \mathcal{D}_n}[|S|] \le n \cdot \frac{1}{\sqrt{n} \cdot \log^2 n} = \frac{\sqrt{n}}{\log^2 n}$. □

Let $x \in \{0,1,\ldots,n-1\}$, and henceforth let $\mathcal{D}_n^{+x}$ be the following distribution: sample from $\mathcal{D}_n$ until we obtain a set $S \ni x$, and output S. Given $x, y \in \{0,1,\ldots,n-1\}$, write x and y in their binary representations, i.e., $x, y \in \{0,1\}^{\log n}$. We say that x and y are *related* or $\mathsf{Related}(x,y) = 1$, if they share a common suffix of length at least $\frac{1}{2}\log n + 1$. Given a set $S \subseteq \{0,1,\ldots,n-1\}$, let $N_{\text{related}}(S,x)$ be the number of elements in S that are related to x.

Lemma 5.2 (Number of related elements in sampled set)**.** *Fix an arbitrary element $x \in \{0,1,\ldots,n-1\}$. Then,*

$$\mathbb{E}_{S \xleftarrow{\$} \mathcal{D}_n^{+x}}\left[N_{\text{related}}(S,x)\right] \le \frac{1}{\log n}$$

Proof. Let $k \in [\frac{1}{2}\log n + 1, \log n - 1]$, there are at least $2^{\log n - k}$ strings in $\{0,1\}^{\log n}$ that share suffix of length at least k as $x \in \{0,1\}^{\log n}$. Let T_k denote the set of $2^{\log n - k}$ strings in $\{0,1\}^{\log n}$ and not equal to x that share a suffix of length at least k as x. Let $T_k^* \subseteq T_k$ be the subset that share a *longest* common suffix of length *exactly* k as x.

For any $y \in T_k^*$, $\Pr_{S \xleftarrow{\$} \mathcal{D}_n^{+x}}[y \in S] = \frac{1}{2^{\log n + B - k}}$. By linearity of expectation,

$$\mathbb{E}_{S \xleftarrow{\$} \mathcal{D}_n^{+x}}[|T_k^* \cap S|] \le \frac{1}{2^{\log n + B - k}} \cdot 2^{\log n - k} = \frac{1}{2^B}$$

Observe that $N_{\text{related}}(S,x) = \sum_{k=\frac{1}{2}\log n+1}^{\log n - 1} |T_k^* \cap S|$. Therefore,

$$\mathbb{E}_{S \xleftarrow{\$} \mathcal{D}_n^{+x}}[N_{\text{related}}(S,x)] \le \frac{1}{2^B} \cdot \left(\frac{1}{2}\log n - 1\right) \le \frac{1}{2^B} \cdot \log n \le \frac{1}{\log n}$$

□

Henceforth let $\mathcal{D}_n^m$ be the distribution where we sample from $\mathcal{D}_n$ independently at random for m number of times.

Lemma 5.3 (Coverage probability)**.** *Let $m \ge 6\sqrt{n} \cdot \log^3 n$. For any fixed $x \in \{0,1,\ldots,n-1\}$,* $\Pr_{S_1,\ldots,S_m \xleftarrow{\$} \mathcal{D}_n^m}[x \notin \cup_{i \in [m]} S_i] \le 1/n$.

Proof. Every element x is included in the set $S \xleftarrow{\$} \mathcal{D}_n$ with probability $\left(\frac{1}{2}\right)^{\log n/2 + B} = \frac{1}{\sqrt{n} \cdot 2^B} \ge \frac{1}{2\sqrt{n} \cdot \log^2 n}$. Therefore, the probability that x does not appear in $\cup_{i \in [m]} S_i$ is

$$\left((1 - \frac{1}{2\sqrt{n} \cdot \log^2 n})^{2\sqrt{n} \cdot \log^2 n}\right)^{3\log n} \le \left(\frac{1}{e}\right)^{3\log n} \le 1/n$$

□

Efficient set enumeration. The following algorithm can efficiently enumerate elements in the sampled set, by making not too many queries to the random oracle RO.

Set Enumeration Algorithm

1. let $i_0 := \frac{1}{2}\log n + 1$, and let $Z_{i_0} := \left\{z \in \{0,1\}^{i_0} : \mathsf{RO}(z) = 1\right\}$;
2. for $i = i_0 - 1$ to $\log n$: let $Z_i := \left\{z \in \{0,1\}^i : \mathsf{RO}(z) = 1 \text{ and } z[2:] \in Z_{i-1}\right\}$ where $z[2:]$ denotes the string z with the first bit removed;
3. let $S := \emptyset$, for every $z \in Z_{\log n}$:

- let $y = 0^B || z$;
- if $\mathsf{RO}(y[i:]) = 1$ for all $i \in [1, B]$, then $S := S \cup \{x\}$;

and output S.

Given $x \in \{0, 1, \ldots, n-1\}$, henceforth we use the notation $\mathsf{RO} \xleftarrow{\$} \mathcal{D}_n^{+x}$ to mean sample the random oracle RO until the set determined by RO contains the element x. We use the notation $\mathsf{EnumTime}(\mathsf{RO})$ to denote the total number of RO calls made by the above set enumeration algorithm to enumerate the set generated by RO.

Lemma 5.4 (Efficient set enumeration)**.** *Suppose that $n \geq 4$. For any fixed $x \in \{0, 1, \ldots, n-1\}$,*

$$\Pr_{\mathsf{RO} \xleftarrow{\$} \mathcal{D}_n^{+x}} \left[\mathsf{EnumTime}(\mathsf{RO}) > 6\sqrt{n} \log^5 n\right] \leq 1/\log n$$

Proof. Henceforth, for every element z ever written down in one of the sets $Z_{i_0}, \ldots, Z_{\log n}$ in the above set enumeration algorithm, we say that z is *eligible*.

In the above set enumeration algorithm, first, we make 2^{i_0} RO calls on all strings of length $i_0 := \frac{1}{2}\log n + 1$. Then, for every eligible element z whose length is smaller than $\log n$, at most 2 RO calls are made, to determine whether $0||z$ and $1||z$ are eligible. Finally, for every eligible string of length exactly $\log n$, $2^B \leq 2\log^2 n$ more additional calls to RO are made. It suffices to prove that except with $1/\log n$ probability, it must be that for all $i \in [i_0, \log n]$, $|Z_i| \leq 2\sqrt{n} \cdot \log^2 n$. If so, the total number of RO calls is upper bounded by $2^{i_0} + \log n \cdot 2\log^2 n \cdot 2\sqrt{n} \cdot \log^2 n \leq 6\sqrt{n}\log^5 n$.

Fix some $i \in [i_0, \log n]$, consider all strings z of length i.

- *Case 1:* the longest common suffix of z and x is shorter than $\frac{1}{2}\log n + 1$. In this case, for any such fixed z, $\Pr_{\mathcal{D}_n^{+x}}[z \in Z_i] = \left(\frac{1}{2}\right)^{i - \frac{1}{2}\log n}$. There are at most 2^i such elements, and thus the expected number of elements from this set that land in Z_i is at most $2^i \cdot \left(\frac{1}{2}\right)^{i - \frac{1}{2}\log n} = 2^{\frac{1}{2}\log n} = \sqrt{n}$.
- *Case 2:* the longest common suffix of z and x if of length exactly $k \geq \frac{1}{2}\log n + 1$. In this case, for any such fixed z, $\Pr_{\mathcal{D}_n^{+x}}[z \in Z_i] = \left(\frac{1}{2}\right)^{i-k}$. There are at most 2^{i-k} such elements, and the expected number of such elements that contribute to Z_i is at most 1 for every fixed k, and the expected number over all k's is at most $\log n$.

Summarizing the above cases, the expected number of strings in Z_i is at most $\sqrt{n} + \log n$; for $n \geq 4$, this is upper bounded by $2\sqrt{n}$. By the Markov Inequality, $\Pr_{\mathcal{D}_n^{+x}}[|Z_i| > 2\sqrt{n}\log^2 n] < 1/\log^2 n$. By the union bound, $\Pr_{\mathcal{D}_n^{+x}}[\forall i \in [i_0, \log n] : |Z_i| \leq 2\sqrt{n}\log^2 n] \geq 1 - 1/\log n$. □

5.3 Construction

Below we present our PRSet construction for the family of distributions $\{\mathcal{D}_n\}_{n \in \mathbb{N}}$ described in Section 5.2.

Privately Puncturable Pseudorandom Set PRSet

Notations. Let $\mathsf{PRF} := (\mathbf{Gen}, \mathbf{Eval}, \mathbf{Puncture}, \mathbf{PEval})$ be a privately puncturable PRF. Let $B := \lceil 2\log\log n \rceil$.

Algorithm.

- **Gen**$(1^\lambda, n)$:
  1. call $\mathsf{sk}' \leftarrow \mathsf{PRF}.\mathbf{Gen}(1^\lambda, L = \log n + B, m = \frac{1}{2}\log n + B)$;
  2. let P be an arbitrary set of m distinct strings in $\{0,1\}^L$ that begin with the bit 1;
  3. call $\mathsf{sk} \leftarrow \mathsf{PRF}.\mathbf{Puncture}(\mathsf{sk}', P)$, and output $(\mathsf{sk}, \mathsf{msk} = \mathsf{sk}')$.
- **Set**(sk):
  1. let $i_0 := \frac{1}{2}\log n + 1$, and let $Z_{i_0} := \left\{z \in \{0,1\}^{i_0} : \mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}, z) = 1\right\}$;
  2. for $i = i_0 - 1$ to $\log n$: let $Z_i := \left\{z \in \{0,1\}^i : \mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}, z) = 1 \text{ and } z[2:] \in Z_{i-1}\right\}$ where $z[2:]$ denotes the string z with the first bit removed;
  3. let $S := \emptyset$, for every $z \in Z_{\log n}$:
     - let $y = 0^B||z$;
     - if $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}, y[i:]) = 1$ for all $i \in [1, B]$, then $S := S \cup \{x\}$;

     and output S.
- **Member**(sk, x):
  1. write $x \in \{0,1\}^{\log n}$ as a binary string, and let $z := 0^B||x$;
  2. if for every $1 \le i \le \frac{1}{2} \cdot \log n + B$, $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}, z[i:]) = 1$ where $z[i:]$ denotes the length-$(\log n + B - i + 1)$ suffix of z, then output 1; else output 0.
- **Puncture**(msk, x):
  1. write $x \in \{0,1\}^{\log n}$ as a binary string, and let $z := 0^B||x$;
  2. let $P := \{z[i:]\}_{i \in [1, \frac{1}{2}\cdot\log n + B]}$ and $\mathsf{sk}_P \leftarrow \mathsf{PRF}.\mathbf{Puncture}(\mathsf{msk}, P)$; output sk_P.

5.4 Proofs

Lemma 5.5 (Correctness, pseudorandomness, and functionality preservation under puncturing)**.** *The above* PRSet *construction satisfies correctness. Further, suppose that the* PRF *scheme satisfies pseudorandomness; then the* PRSet *scheme also satisfies pseudorandomness and functionality preservation under punturing.*

Proof. Correctness follows directly from the construction. Pseudorandomness relies on the pseudorandomness of the PRF through a straightforward reduction. To see functionality preservation, let $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$, let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$, and below we may ignore the negligible probability event that the underlying puncturable PRF violates its functionality preservation property. Notice that for every string z that is a suffix of $0^B||x$ of length at least $\frac{1}{2}\log n + 1$, $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}, z) = 1$, but there may exist such z where $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}_x, z)$ becomes 0 instead. For any string z that is not a suffix of $0^B||x$ of length at least $\frac{1}{2}\log n + 1$, $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}, z) = \mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}_x, z)$. Given the above observation, "functional preservation under puncturing" is easy to verify. □

Lemma 5.6 (Security w.r.t. puncturing)**.** *Suppose that the* PRF *scheme satisfies pseudorandomness and privacy w.r.t. puncturing as defined in Section 4. Then, the above* PRSet *construction satisfies security w.r.t. puncturing.*

Proof. We need to prove two properties.

First property. We begin by proving the first propety, that is, the following distributions are computationally indistinguishable for any $x \in \{0, 1, \ldots, n-1\}$:

- Expt_0: Repeat $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $x \in \mathbf{Set}(\mathsf{sk})$, let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$, and output sk_x.
- Expt_1: $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ and output sk.

We define an intermediate hybrid experiment Hyb: sample $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$, let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$, and output sk_x.

Claim 5.7. *Suppose that the puncturable* PRF *scheme satisfies pseudorandomness as defined in Section 4. Then,* Expt_0 *and* Hyb *are computationally indistinguishable.*

Proof. Suppose that there is an efficient adversary $\mathcal{A}$ that can distinguish Expt_0 and Hyb with non-negligible probability. We can construct an efficient reduction $\mathcal{B}$ that breaks the pseudorandomness of the PRF scheme.

Henceforth let P_x denote the set containing the $m = \frac{1}{2}\log n + B$ queries we need to make to determine whether x is in the set. $\mathcal{B}$ asks its own challenger denoted $\mathcal{C}$ for a PRF key punctured at P_x, and it obtains sk_{P_x}. It forwards sk_{P_x} to $\mathcal{A}$. $\mathcal{B}$ then obtains a vector of bits denoted $\boldsymbol{\beta} := (\beta_1, \ldots, \beta_m)$ as the purported outcomes for $\{\mathbf{Eval}(\mathsf{sk}, y)\}_{y \in P_x}$. If $\boldsymbol{\beta} = \mathbf{1}$, then $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs. Else, it outputs a random bit.

Case 1. If the challenger $\mathcal{C}$ is using real values for $\{\mathbf{Eval}(\mathsf{sk}, y)\}_{y \in P_x}$, then $\mathcal{A}$'s view is identically distributed as Expt_0. In this case, the probability that $\mathcal{B}$ outputs 1 is equal to

$$p := \Pr[\mathcal{A}(\mathsf{Expt}_0) = 1] \cdot \Pr[\boldsymbol{\beta} = \mathbf{1}] + \frac{1}{2} \cdot \Pr[\boldsymbol{\beta} \neq \mathbf{1}]$$

Case 2. If the challenger $\mathcal{C}$ is using random values in place of $\{\mathbf{Eval}(\mathsf{sk}, y)\}_{y \in P_x}$, then $\mathcal{A}$'s view is identically distributed as Hyb. In this case, the probability that $\mathcal{B}$ outputs 1 is equal to

$$p' := \Pr[\mathcal{A}(\mathsf{Hyb}) = 1] \cdot \Pr[\boldsymbol{\beta} = \mathbf{1}] + \frac{1}{2} \cdot \Pr[\boldsymbol{\beta} \neq \mathbf{1}]$$

Note that in Case 1, $|\Pr[\boldsymbol{\beta} = \mathbf{1}] - 1/2^m| \leq \mathsf{negl}(\lambda)$ due to the pseudorandomness of the PRF; and in Case 2 $\Pr[\boldsymbol{\beta} = \mathbf{1}] = 1/2^m$. Moreover, $1/2^m$ is non-negligible due to the choice of m. Therefore, if $|\Pr[\mathcal{A}(\mathsf{Expt}_0) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}) = 1]|$ is non-negligible, then $|p - p'|$ would be non-negligible, too. □

Claim 5.8. *Suppose that the puncturable* PRF *scheme satisfies privacy w.r.t. puncturing as defined in Section 4. Then,* Hyb *is computationally indistinguishable from* Expt_1.

Proof. Follows from a straightforward reduction to the privacy w.r.t. puncturing property of the PRF. □

The computational indistinguishability of Expt_0 and Expt_1 now follows from Claim 5.7 and Claim 5.8.

Second property. We next prove the second property, that is, we want to show that the following two distributions are computationally indistinguishable:

- Expt_0^*: Repeat $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $x \in \mathbf{Set}(\mathsf{sk})$, let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$, and output $(\mathbf{Set}(\mathsf{sk}), x \in \mathbf{Set}(\mathsf{sk}_x))$.
- Expt_1^*: Repeat $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $x \in \mathbf{Set}(\mathsf{sk})$, and output $(\mathbf{Set}(\mathsf{sk}), \mathsf{Bernoulli}(\rho))$ where $\rho := 2^{-(\frac{1}{2}\log n + B)}$.

Let $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $x \in \mathbf{Set}(\mathsf{sk})$, and let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$. Observe that there is a deterministic, polynomial-time function **Reconstruct** such that $\mathbf{Reconstruct}(\mathsf{sk}_x, x) = \mathbf{Set}(\mathsf{sk})$. Essentially **Reconstruct** uses answers to $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}_x, \cdot)$ calls to determine set membership, except that when encountering any string z that is a suffix of $0^B||x$ of length at least $\frac{1}{2}\log n + 1$, override the outcome of $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}_x, z)$ to 1.

We can therefore rewrite Expt_0^* as the following experiment Hyb: repeat $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathbf{Gen}(1^\lambda, n)$ until $x \in \mathbf{Set}(\mathsf{sk})$, let $\mathsf{sk}_x \leftarrow \mathbf{Puncture}(\mathsf{msk}, x)$, and output $(\mathbf{Reconstruct}(\mathsf{sk}_x, x), x \in \mathbf{Set}(\mathsf{sk}_x))$.

Due to the first property which we just proved, the above distribution Hyb is computationally indistinguishable from the following Hyb': $(\mathsf{sk}, \_) \leftarrow \mathbf{Gen}(1^\lambda, n)$, and output $(\mathbf{Reconstruct}(\mathsf{sk}, x), x \in \mathbf{Set}(\mathsf{sk}))$.

Now, consider the experiment Ideal which is defined just like in Hyb, except that sampling a PRF secret key is replaced with sampling an RO, and to determine set membership, any call to $\mathsf{PRF}.\mathbf{PEval}(\mathsf{sk}, \cdot)$ is replaced with $\mathsf{RO}(\cdot)$. In Ideal, observe that $\mathbf{Reconstruct}(\mathsf{RO}, x)$ does not need to look at the coins that determine x's membership in the set. Based on this observation as well as the pseudorandomness of the underlying PRF, we conclude that Ideal is computationally indistinguishable from Expt_1^*.

□

6 Definitions: Two-Server preprocessing PIR

We first define a two-server preprocessing PIR scheme that supports multiple queries. Henceforth, the two servers is treated as stateful algorithms Left and Right, respectively, and the client is treated as a stateful algorithm denoted Client. Initially, all of Client, Left, and Right receive the parameters 1^λ and n.

- **Offline setup.** Client receives nothing and each of Left and Right receives the same database $\mathsf{DB} \in \{0,1\}^n$ as input. Client sends a single message to Left, and Left responds with a single message often called a *hint*.
- **Online queries.** The following can be repeated for apriori-unknown polynomially many steps. Upon receiving an index $x \in \{0, 1, \ldots, n-1\}$ to query, Client sends a single message to Left and a single message to Right. It receives a single response from each server Left and Right. Client then performs some computation and outputs an answer $\beta \in \{0,1\}$.

Correctness. Given a database $\mathsf{DB} \in \{0,1\}^n$ where the bits are indexed $0, 1, \ldots, n-1$, respectively, the correct answer for a query $x \in \{0, 1, \ldots, n-1\}$ is the x-th bit of DB.

For correctness, we require that for any Q, n that are polynomially bounded in λ, there is a negligible function $\mathsf{negl}(\cdot)$, such that for any database $\mathsf{DB} := \{0,1\}^n$, for any sequence of queries $x_1, x_2, \ldots, x_Q \in \{0, 1, \ldots, n-1\}$, the probability that an honest execution of the offline/online PIR scheme with DB and queries $x_1, x_2, \ldots, x_Q$ returns all correct answers with probability $1 - \mathsf{negl}(\lambda)$.

Privacy. For privacy, we require the following.

- *Left-server privacy.* We want that there is a p.p.t. stateful simulator Sim, such that for any arbitrary (even computionally unbounded) algorithm Right^*, for any non-uniform p.p.t. adversary $\mathcal{A}$, $\mathcal{A}$'s views in the following Real and Ideal experiments are computationally indistinguishable:

  1. Real: The honest Client interacts with $\mathcal{A}$ who acts as the left server and may deviate arbitrarily from the prescribed protocol, and an arbitrary (even computionally unbounded) algorithm Right^* acting as the right server. In every online step t, $\mathcal{A}$ adaptively chooses the next query $x_t \in \{0, 1, \ldots, n-1\}$, and Client is invoked with x_t.
  2. Ideal: The simulated client Sim interacts with $\mathcal{A}$ who acts as the left server, and an arbitrary (even computionally unbounded) algorithm Right^* acting as the right server. In every online step t, $\mathcal{A}$ adaptively chooses the next query $x_t \in \{0, 1, \ldots, n-1\}$, and Sim is invoked without receiving x_t.

- *Right-server privacy.* Right-server privacy is defined in a symmetric way as above by exchanging left and right.

Intuitively, the above privacy definition requires that any single server alone cannot learn anything about the client's queries through its interactions with the client; further, this must hold even when both servers can behave maliciously. However, recall that we do not guarantee correctness if one or both server(s) fail to respond correctly.

7 Warmup: Single-Copy Scheme

7.1 Construction

We present the single-copy scheme below.

Warmup: Single-Copy Scheme PIR^1

Offline setup. For $i = 1$ to $\mathsf{lenT} := 6\sqrt{n} \cdot \log^3 n$ in parallel:

1. Client: Sample $(\mathsf{sk}_i, \mathsf{msk}_i) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$, send sk_i to Left.
2. Left: Run $S_i \leftarrow \mathsf{PRSet.Set}(\mathsf{sk}_i)$. If the runtime of $\mathsf{PRSet.Set}(\mathsf{sk}_i)$, measured in terms of $\mathsf{PRF.PEval}$ calls, exceeds $\mathsf{maxT} := 6\sqrt{n}\log^5 n$, return $p_i := 0$ to Client. Else, return the parity bit $p_i \in \{0, 1\}$ of the set S_i to Client.
3. Client: Save $T_i := (\mathsf{sk}_i, \mathsf{msk}_i, p_i)$ where $T := (T_1, T_2, \ldots, T_{\mathsf{lenT}})$ denotes a table saved by Client.

Note that if we run the above in parallel for all indices i, the offline phase consists of a single message from Client to Left and a single response back.

Online query for index $x \in \{0, 1, \ldots, n-1\}$**.**

- **Query** ($\mathsf{Client} \Leftrightarrow \mathsf{Right}$):

  1. Client:

- Sample a $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathsf{PRSet.}\mathbf{Gen}(1^\lambda, n)$ subject to $\mathsf{PRSet.}\mathbf{Member}(\mathsf{sk}, x) = 1$, append $(\mathsf{sk}, \mathsf{msk}, 0)$ to the end of the table T.
- Henceforth parse $T_i := (\mathsf{sk}_i, \mathsf{msk}_i, p_i)$. Let j be the smallest entry in the table T such that $\mathsf{PRSet.}\mathbf{Member}(\mathsf{sk}_j, x) = 1$.
- Call $\widetilde{\mathsf{sk}}_j := \mathsf{PRSet.}\mathbf{Puncture}(\mathsf{msk}_j, x)$. Send $\widetilde{\mathsf{sk}}_j$ to Right.

2. Right: Run $S \leftarrow \mathsf{PRSet.}\mathbf{Set}(\widetilde{\mathsf{sk}}_j)$. If the runtime exceeds maxT, return $p := 0$ to Client. Else, return the parity bit $p \in \{0, 1\}$ of the set S to Client.
3. Client: Let $\beta' := p \oplus p_j$ be a candidate answer. Obtain the true answer β from an oracle[a].

- **Refresh** (Client $\Leftrightarrow$ Left):

1. Client:
   - Sample a new $(\mathsf{sk}', \mathsf{msk}') \leftarrow \mathsf{PRSet.}\mathbf{Gen}(1^\lambda, n)$ subject to $\mathsf{PRSet.}\mathbf{Member}(\mathsf{sk}', x) = 1$.
   - Call $\mathsf{sk}'_x \leftarrow \mathsf{PRSet.}\mathbf{Puncture}(\mathsf{msk}', x)$, and send sk'_x to Left.
2. Left: Run $S \leftarrow \mathsf{PRSet.}\mathbf{Set}(\mathsf{sk}'_x)$. If the runtime exceeds maxT, return $p := 0$ to Client. Else, return the parity bit $p \in \{0, 1\}$ of the set S to Client.
3. Client: Replace $T_j := (\mathsf{sk}', \mathsf{msk}', p \oplus \beta)$. Finally, remove the last entry from the table T.

[a] We will instantiate this true-answer oracle in our final multi-copy scheme in Section 8, simply by taking majority vote among all copies, which is guaranteed to be correct with all but negligible probability.

Fact 7.1. *In the above PIR scheme, all messages sent by the client to the servers depend only on the clients' local randomness and the sequence of queries so far, but do not depend on any past message received from the server.*

7.2 Proofs of Privacy

Theorem 7.2 (Left-server privacy)**.** *Suppose that the* PRSet *scheme satisfies (the first property in) "security w.r.t. puncturing". Then, the single-copy scheme in Section 7.1 satisfies left-server privacy.*

Proof. We define the following simulator Sim which fully specifies the ideal experiment Ideal:

- *Offline setup.* For $i = 1$ to $\mathsf{lenT} := 6\sqrt{n} \cdot \log^3 n$: sample $(\mathsf{sk}_i, \mathsf{msk}_i) \leftarrow \mathsf{PRSet.}\mathbf{Gen}(1^\lambda, n)$ and send $\{\mathsf{sk}_i\}_{i \in [1, \mathsf{lenT}]}$ to $\mathcal{A}$ acting as the left server.
- *Online queries.* For each online query, sample $(\mathsf{sk}', \mathsf{msk}') \leftarrow \mathsf{PRSet.}\mathbf{Gen}(1^\lambda, n)$ and send sk' to $\mathcal{A}$ acting as the left server.

The computational indistinguishability of $\mathcal{A}$'s views in Real and Ideal follow due to a straightforward hybrid argument relying on the "security w.r.t. puncturing" property of the PRSet scheme. Specifically, let Q be the total number of queries in the online phase. We define a sequence of hybrid experiments $\{\mathsf{Hyb}_i\}_{i \in \{0,1,\ldots,Q\}}$, where in Hyb_i, during the first i online steps, $\mathcal{A}$ (acting as the left server) receives a message constructed like in Ideal, and during the remaining $Q - i$ online steps, $\mathcal{A}$ receives a message constructed like in Real. Clearly, $\mathsf{Hyb}_0 = \mathsf{Real}$ and $\mathsf{Hyb}_Q = \mathsf{Ideal}$. It suffices to show that any two adjacent hybrid experiments are computationally indistinguishable, and this follows due to a straightforward reduction to the "security w.r.t. puncturing" property of the PRSet scheme. □

Theorem 7.3 (Right-server privacy)**.** *Suppose that the* PRSet *scheme satisfies (the first property in) security w.r.t. puncturing. Then, the single-copy scheme in Section 7.1 satisfies right-server privacy.*

Proof. We define the following simulator Sim which fully specifies the ideal experiment Ideal:

- *Offline setup.* $\mathcal{A}$, acting as the right server, receives nothing.
- *Online queries.* For each online query, sample $(\mathsf{sk}', \mathsf{msk}') \leftarrow \mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)$ and send sk' to $\mathcal{A}$ acting as the right server.

We now need to argue that any non-uniform p.p.t.$\mathcal{A}$'s views in Real and Ideal are computationally indistinguishable.

Real^*. First, we consider the following experiment Real^*.

- *Offline setup.* For each $i \in [1, \mathsf{lenT}]$, Client samples $(\mathsf{sk}_i, \mathsf{msk}_i) \leftarrow \mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)$, and lets $T_i := (\mathsf{sk}_i, \mathsf{msk}_i)$. The adversary $\mathcal{A}$, acting as the right server, receives nothing.
- *Online queries.* For each online query $x \in \{0, 1, \ldots, n-1\}$:
  a) Client samples $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)$ subject to $\mathsf{PRSet}.\mathbf{Member}(\mathsf{sk}, x) = 1$, and appends $(\mathsf{sk}, \mathsf{msk})$ to the end of the table T.
  b) Client finds the smallest entry $T_j := (\mathsf{sk}_j, \mathsf{msk}_j)$ in T such that $\mathsf{PRSet}.\mathbf{Member}(\mathsf{sk}_j, x) = 1$. It sends $\mathsf{PRSet}.\mathbf{Puncture}(\mathsf{msk}_j, x)$ to $\mathcal{A}$ acting as the right server.
  c) Client samples $(\mathsf{sk}', \mathsf{msk}') \leftarrow \mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)$ subject to $\mathsf{PRSet}.\mathbf{Member}(\mathsf{sk}, x) = 1$, overwrites T_j with $(\mathsf{sk}', \mathsf{msk}')$, and removes the last entry from T.

Real^* is just a rewrite of Real throwing away terms that we do not care. Let $\mathsf{View}^{\mathsf{Real}}$ and $\mathsf{View}^{\mathsf{Real}^*}$ denote $\mathcal{A}$'s view and the client's table T (truncating the third field of each entry in Real) at the beginning of each online query, in the experiments Real and Real^*, respectively. We have that even for a computationally unbounded $\mathcal{A}$, $\mathsf{View}^{\mathsf{Real}}$ and $\mathsf{View}^{\mathsf{Real}^*}$ are identically distributed.

Fact 7.4. *In* Real^**, for every online step* t*, even if* $\mathcal{A}$ *is computationally unbounded, and even when conditioned on* $\mathcal{A}$*'s view over the first* $t-1$ *steps,*

- *let* $x \in \{0, 1, \ldots, n-1\}$ *be the* t*-th online query, the message* $\mathcal{A}$ *receives in the* t*-th query is distributed as: sample* $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)$ *subject to* $\mathsf{PRSet}.\mathbf{Member}(\mathsf{sk}, x) = 1$ *and output* $\mathsf{PRSet}.\mathbf{Puncture}(\mathsf{sk}, x)$*;*
- *at the end of the* t*-th online query, the client's table* T *is a fresh uniform sample from* $\mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)^{\mathsf{lenT}}$ *independent of the message* $\mathcal{A}$ *receives during the* t*-th query, i.e.,* T *contains a sample of* lenT *uniform, independent entries from the distribution* $\mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)$*.*

Proof. We can prove by induction.

Base case. At the end of the offline phase (henceforth also called the 0-th query), indeed the client's table T is uniform sample from the distribution $\mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)^{\mathsf{lenT}}$.

Inductive step. Suppose that at the end of the t-th step, the client's table T is uniform sample from the distribution $\mathsf{PRSet.Gen}(1^\lambda, n)^{\mathsf{lenT}}$ even when conditioned on $\mathcal{A}$'s view in the first t steps. We now prove that the stated claims hold for $t+1$.

Let $x \in \{0, 1, \ldots, n-1\}$ be the query made in online step $t+1$, the choice of x depends only on $\mathcal{A}$'s view in the first t online queries. Henceforth, for $i \in [1, \mathsf{lenT}]$, let $\alpha_{i,x}$ be the probability that in a random sample from the distribution $\mathsf{PRSet.Gen}(1^\lambda, n)^{\mathsf{lenT}}$, the first entry that contains x is i. Let $\alpha_{\mathsf{lenT}+1,x} := 1 - \sum_{i=1}^{\mathsf{lenT}} \alpha_{i,x}$.

Consider the following experiment Expt:

- Client samples an index $u \in [\mathsf{lenT}+1]$ such that $u = i$ with probability $\alpha_{i,x}$.
- $\forall j < u$, Client samples $T_j := (\mathsf{sk}_j, \mathsf{msk}_j) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$ subject to $\mathsf{PRSet.Member}(\mathsf{sk}_j, x) = 0$.
- For u, Client samples $(\mathsf{sk}, \mathsf{msk})$ and $(\mathsf{sk}', \mathsf{msk}')$ independently from the distribution $\mathsf{PRSet.Gen}(1^\lambda, n)$ subject to $\mathsf{PRSet.Member}(\mathsf{sk}_j, x) = 1$. It sends $\mathsf{PRSet.Puncture}(\mathsf{sk}, x)$ to $\mathcal{A}$ and saves $T_u := (\mathsf{sk}', \mathsf{msk}')$.
- $\forall j \in [u+1, \mathsf{lenT}+1]$, Client samples $T_j := (\mathsf{sk}_j, \mathsf{msk}_j) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$.
- Finally, Client removes last entry from T.

Let $\mathsf{View}_{t+1}^{\mathsf{Real}^*}$ be the message $\mathcal{A}$ receives during the $(t+1)$-st query as well as the client's table T at the end of the $(t+1)$-st query in Real^*. Let $\mathsf{View}^{\mathsf{Expt}}$ be the message $\mathcal{A}$ receives as well as the client's table T at the end in Expt. Suppose that the induction hypothesis holds, then it is not hard to see that $\mathsf{View}^{\mathsf{Expt}}$ is identically distributed as $\mathsf{View}_{t+1}^{\mathsf{Real}^*}$ even when conditioning on the view of $\mathcal{A}$ in the first t queries in Real^*, and even when $\mathcal{A}$ is computationally unbounded.

In the experiment Expt, it is not hard to see the distribution $\mathsf{View}^{\mathsf{Expt}}$ is the following: T is sampled at random from $\mathsf{PRSet.Gen}(1^\lambda, n)^{\mathsf{lenT}}$, and $\mathcal{A}$'s received message is distributed as: sample $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$ subject to $\mathsf{PRSet.Member}(\mathsf{sk}, x) = 1$ and output $\mathsf{PRSet.Puncture}(\mathsf{sk}, x)$. □

Given Fact 7.4, we can prove that any non-uniform p.p.t. $\mathcal{A}$'s views in Ideal and Real^* are computationally indistinguishable through a standard hybrid argument, relying on the the "security w.r.t. puncturing" property of the PRSet scheme — the hybrid sequence is similar to the proof of Theorem 7.2. □

7.3 Occasional Correctness

Experiment CExpt. We consider the following experiment CExpt.

Correctness Experiment CExpt

Offline setup. For $j = 1$ to $\mathsf{lenT} := 6\sqrt{n} \cdot \log^3 n$: sample $(\mathsf{sk}_j, \mathsf{msk}_j) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$; let $T_j := (\mathsf{sk}_j, \mathsf{msk}_j)$, and set $\mathsf{label}(T_j) := \perp$.

Online query for index $x \in \{0, 1, \ldots, n-1\}$.

a) Sample $(\mathsf{sk}^*, \mathsf{msk}^*) \xleftarrow{\$} \mathsf{PRSet}(1^\lambda, n)$ subject to $x \in \mathsf{PRSet.Set}(\mathsf{sk}^*)$. Append $(\mathsf{sk}^*, \mathsf{msk}^*)$ to the table T as the last entry, and mark its label $\mathsf{label}(T_{\mathsf{lenT}+1}) := \perp$.

b) Let $T_j := (\mathsf{sk}_j, \mathsf{msk}_j)$ be the smallest entry in the table T such that $x \in \mathsf{PRSet.Set}(\mathsf{sk}_j)$.

c) If $z := \mathsf{label}(T_j) \neq \perp$ and it is not the case that $\mathbf{Set}(\mathsf{sk}_{j,z}) = \mathbf{Set}(\mathsf{sk}_j) \backslash \{z\}$ where $\mathsf{sk}_{j,z} \leftarrow \mathsf{PRSet}.\mathbf{Puncture}(\mathsf{msk}_j, z)$, then return Err-PunctureLeft.

d) If it is not the case that $\mathbf{Set}(\mathsf{sk}_{j,x}) = \mathbf{Set}(\mathsf{sk}_j) \backslash \{x\}$ where $\mathsf{sk}_{j,x} \leftarrow \mathsf{PRSet}.\mathbf{Puncture}(\mathsf{msk}_j, x)$, then return Err-PunctureRight.

e) If $\mathsf{PRSet}.\mathbf{Set}(\mathsf{sk}_j)$ runs in time more than maxT, return Err-ExceedTime.

f) Sample a new $(\mathsf{sk}', \mathsf{msk}') \leftarrow \mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)$ subject to $x \in \mathsf{PRSet}.\mathbf{Set}(\mathsf{sk}')$. Overwrite $T_j := (\mathsf{sk}', \mathsf{msk}')$ and set $\mathsf{label}(T_j) := x$.

g) If $j = \mathsf{lenT} + 1$, return Err-NotFound.

h) Remove the last entry from T, and return Success.

Let $\mathbf{Wrong}^{i,\mathsf{CExpt}}(x_1, \ldots, x_Q)$ be the event that the i-th query returns an error message in CExpt with the query sequence $x_1, \ldots, x_Q$. Let $\mathbf{Wrong}^{i,\mathsf{Real}}(\mathsf{DB}, x_1, \ldots, x_Q)$ be the event that during the i-th query, the candidate answer β' is incorrect during an honest execution of the single-copy scheme in Section 7, using DB and queries $x_1, \ldots, x_Q$ as input.

Fact 7.5. *Suppose that the* PRSet *scheme satisfies "functional preservation under puncturing". Then, for any* $\mathsf{DB} \in \{0,1\}^n$*, for any* $Q \in \mathbb{N}$ *that is polynomially bounded in* λ*, for any sequence of queries* $x_1, x_2, \ldots, x_Q \in \{0, 1, \ldots, n-1\}$*, for any* $i \in [Q]$, $\Pr[\mathbf{Wrong}^{i,\mathsf{CExpt}}(x_1, \ldots, x_Q)] \geq \Pr[\mathbf{Wrong}^{i,\mathsf{Real}}(\mathsf{DB}, x_1, \ldots, x_Q)] - \mathsf{negl}(\lambda)$.

Proof. In an honest execution of the single-copy scheme of Section 7, correctness is independent of DB. Henceforth we may assume that $\mathsf{PRSet}.\mathbf{Puncture}$ never violates the "functionality preservation under puncturing" property. Since this bad event happens only with negligible probability, technically, ignoring the bad event translates to applying a union bound and subtracting the negligible probability at the end — this explains the $\mathsf{negl}(\lambda)$ discount in the statement of the fact. Assuming that $\mathsf{PRSet}.\mathbf{Puncture}$ always satisfies the "functionality preservation under puncturing" property, then CExpt is basically a rewrite of the honest execution of the single-copy scheme, with the following modifications all of which either do not affect correctness, or will only increase the chance of incorrectness (in a statistical dominance sense):

1. Remove instructions not related to dertermining correctness;
2. Replace any occurrence of $\mathsf{PRSet}.\mathbf{Member}(\mathsf{sk}, x)$ with the functionally equivalent instruction $x \in \mathsf{PRSet}.\mathbf{Set}(\mathsf{sk})$;
3. Defer checking whether the client knows the correct parity bit corresponding to each table entry T_j to when T_j is consumed during an online query. Note that the client may fail to know the correct parity bit for T_j only if one of the following happens: either the table entry T_j was updated when z was queried, but the set generated by T_j contains elements related to z; or when T_j was updated, the left server exceeded runtime.
4. For an $(\mathsf{sk}, \mathsf{msk})$ in the client's table T, in an honest execution, the client might send a punctured set of T to both the left and right servers, and if either server exceeded runtime, the entry sk might result in an incorrect answer when it is consumed during an online query. In CExpt, due to the "functional preservation under puncturing" property of PRSet, we may use the runtime of $\mathsf{PRSet}.\mathbf{Set}(\mathsf{sk})$ as an over-estimate of the set enumeration time of the left server and right server upon receiving a punctured key, i.e., $\mathsf{sk}_x := \mathsf{PRSet}.\mathbf{Puncture}(\mathsf{msk}, x)$ for some x.

□

Fact 7.6. *In* CExpt*, for any $Q \in \mathbb{N}$ and any sequence of queries $x_1, \ldots, x_Q$, for every $i \in [Q]$, at the end of the i-th online query, the client's table T (not including the labels on entries) is distributed as a fresh random sample from* $\mathsf{PRSet.Gen}(1^\lambda, n)^{\mathsf{lenT}}$.

Proof. The proof is almost the same as that of Fact 7.4. □

Theorem 7.7 (Occasional correctness of the single-copy scheme)**.** *Suppose that the* PRSet *scheme satisfies (the second property in) security w.r.t. puncturing, as well as pseudorandomness. For any $\mathsf{DB} \in \{0,1\}^n$, for any Q that is polynomially bounded in λ, for any sequence of queries $x_1, x_2, \ldots, x_Q \in \{0, 1, \ldots, n-1\}$, for any $i \in [Q]$,*

$$\Pr[\mathbf{Wrong}^{i,\mathsf{Real}}(\mathsf{DB}, x_1, \ldots, x_Q)] \leq 1/3.$$

Proof. Due to Facts 7.5, it suffices to prove that for any $Q \in \mathbb{N}$, for any sequence of queries $x_1, x_2, \ldots, x_Q$, for any $i \in [Q]$, $\Pr[\mathbf{Wrong}^{i,\mathsf{CExpt}}(x_1, \ldots, x_Q)] \leq 1/4$.

Henceforth we focus on CExpt, and we fix an arbitrary fixed sequence of queries $x_1, x_2, \ldots, x_Q$ and an index $i \in [Q]$. Observe that $\mathbf{Wrong}^{i,\mathsf{CExpt}}$ can only occur if the i-th online query returns one of the following error messages {Err-NotFound, Err-ExceedTime, Err-PunctureRight, Err-PunctureLeft }. Therefore, it suffices to show that each of these errors happens with probability at most $3/\log n + \mathsf{negl}(\lambda)$. We upper bound the probability of Err-NotFound, Err-ExceedTime, and Err-PunctureRight in Section 7.3.1, and the probability of Err-PunctureLeft in Section 7.3.2. □

7.3.1 Bounding the Probability of Err-NotFound, Err-ExceedTime, and Err-PunctureRight

Recall that we fix an arbitrary $Q \in \mathbb{N}$, an arbitrary sequence of queries $x_1, x_2, \ldots, x_Q$, an arbitrary $i \in [Q]$. We now bound the probability that the i-th query outputs the error messages Err-NotFound, Err-ExceedTime, and Err-PunctureRight, respectively.

Error Err-NotFound**.** Due to Fact 7.6, and the pseudorandomness of the PRSet scheme,

$$\Pr[\mathsf{Err\text{-}NotFound}] \leq \Pr_{\{S_k\}_{k\in[\mathsf{lenT}]} \xleftarrow{\$} \mathcal{D}_n^{\mathsf{lenT}}}[x_i \notin \cup_{k\in[\mathsf{lenT}]} S_k] + \mathsf{negl}(\lambda)$$

By Lemma 5.3, $\Pr[\mathsf{Err\text{-}NotFound}] \leq 1/n + \mathsf{negl}(\lambda)$.

Error Err-PunctureRight**.** Let $(\mathsf{sk}_j, \mathsf{msk}_j)$ be the entry in the table T matched during the i-th query, such that $x_i \in \mathbf{Set}(\mathsf{sk}_j)$. Recall that Err-PunctureRight happens if it is not the case that $\mathbf{Set}(\mathsf{sk}_{j,x_i}) = \mathbf{Set}(\mathsf{sk}_j) \backslash \{x_i\}$ where $\mathsf{sk}_{j,x_i} \leftarrow \mathsf{PRSet.Puncture}(\mathsf{msk}_j, x_i)$. Henceforth, let Err-RelatedRight be the bad event that $\exists y \in \mathbf{Set}(\mathsf{sk}_j)$, such that $\mathsf{Related}(x_i, y) = 1$; and let Err-RemoveFailRight be the bad event that $x_i \in \mathbf{Set}(\mathsf{sk}_{j,x_i})$. Clearly,

$$\Pr[\mathsf{Err\text{-}PunctureRight}] \leq \Pr[\mathsf{Err\text{-}RelatedRight}] + \Pr[\mathsf{Err\text{-}RemoveFailRight}]$$

Below we will bound $\Pr[\mathsf{Err\text{-}RemoveFailRight}]$ and $\Pr[\mathsf{Err\text{-}RelatedRight}]$ separately. Due to Fact 7.6, during the i-th query, the entry $(\mathsf{sk}_j, \mathsf{msk}_j)$ found is distributed as sampling at random from $(\mathsf{sk}, \mathsf{msk}) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$ subject to $x_i \in \mathbf{Set}(\mathsf{sk})$. Due to the "security w.r.t. puncturing" property of PRSet, sk_{j,x_i} is computationally indistinguishable from a freshly sampled secret key from $\mathsf{PRSet.Gen}(1^\lambda, n)$. Therefore, we have the following:

- Err-RemoveFailRight: $\Pr[\mathsf{Err\text{-}RemoveFailRight}] \leq \Pr_{\mathsf{sk} \leftarrow \mathsf{PRSet}.\mathbf{Gen}(1^\lambda, n)}[x \in \mathbf{Set}(\mathsf{sk})] + \mathsf{negl}(\lambda)$. Due to the pseudorandomness of PRSet, the above is upper bounded by $\Pr_{S \xleftarrow{\$} \mathcal{D}_n}[x \in S] + \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda) \leq \frac{1}{\sqrt{n} \cdot \log^2 n} + \mathsf{negl}(\lambda)$. Therefore, we have that $\Pr[\mathsf{Err\text{-}RemoveFailRight}] \leq \frac{1}{\sqrt{n} \cdot \log^2 n} + \mathsf{negl}(\lambda)$.
- Err-RelatedRight: Due to the pseudorandomness of PRSet, $\Pr[\mathsf{Err\text{-}RelatedRight}] \leq \Pr_{S \leftarrow D_n^{+x_i}}[\exists y \in S : \mathsf{Related}(x_i, y) = 1] + \mathsf{negl}(\lambda)$. Recall that the distribution $D_n^{+x_i}$ means sampling at random from $\mathcal{D}_n$ such that the resulting set contains x_i. Due to Lemma 5.2, it must be that $\Pr_{S \leftarrow D_n^{+x_i}}[\exists y \in S : \mathsf{Related}(x_i, y) = 1] \leq 1/\log n$. Therefore, we have that $\Pr[\mathsf{Err\text{-}RelatedRight}] \leq 1/\log n + \mathsf{negl}(\lambda)$.

Error Err-ExceedTime**.** Due to Fact 7.6 and the pseudorandomness of our PRSet scheme,

$$\Pr[\mathsf{Err\text{-}ExceedTime}] \leq \Pr_{\mathsf{RO} \xleftarrow{\$} D_n^{+x_i}}[\mathsf{EnumTime}(\mathsf{RO}) > \mathsf{maxT}] + \mathsf{negl}(\lambda)$$

By Lemma 5.4, we have that $\Pr_{\mathsf{RO} \xleftarrow{\$} D_n^{+x_i}}[\mathsf{EnumTime}(\mathsf{RO}) > \mathsf{maxT}] \leq 1/\log n$. Hence, $\Pr[\mathsf{Err\text{-}ExceedTime}] \leq 1/\log n + \mathsf{negl}(\lambda)$.

7.3.2 Bounding the Probability of Err-PunctureLeft

Recall that we fix an arbitrary $Q \in \mathbb{N}$, an arbitrary sequence of queries $x_1, x_2, \ldots, x_Q$, an arbitrary $i \in [Q]$. We now bound the probability that the i-th query outputs the error message Err-PunctureLeft.

Let $T_j := (\mathsf{sk}_j, \mathsf{msk}_j)$ be the smallest entry whose corresponding set contains x_i during the i-th query, and define the random variable $y := \mathsf{label}(T_j)$. Let Err-RelatedLeft be the bad event that $\exists z \in \mathbf{Set}(\mathsf{sk}_j)$, such that $\mathsf{Related}(y, z) = 1$; and let Err-RemoveFailLeft be the bad event that $y \in \mathbf{Set}(\mathsf{sk}_{j,y})$ where $\mathsf{sk}_{j,y} \leftarrow \mathsf{PRSet}.\mathbf{Puncture}(\mathsf{msk}_j, y)$. Clearly,

$$\Pr[\mathsf{Err\text{-}PunctureLeft}] \leq \Pr[\mathsf{Err\text{-}RemoveFailLeft}] + \Pr[\mathsf{Err\text{-}RelatedLeft}]$$

Below we upper bound $\Pr[\mathsf{Err\text{-}RemoveFailLeft}]$ and $\Pr[\mathsf{Err\text{-}RelatedLeft}]$ separately.

First, we shall bound $\Pr[\mathsf{Err\text{-}RelatedLeft}]$ and prove the following lemma.

Lemma 7.8. *Suppose that* PRSet *satisfies pseudorandomness. Then,* $\Pr[\mathsf{Err\text{-}RelatedLeft}] \leq 2/\log n + \mathsf{negl}(\lambda)$ *in* CExpt.

Proof. Observe that

$$\mathsf{Err\text{-}RelatedLeft} = ((\mathsf{Related}(x_i, y) = 1) \wedge \mathsf{Err\text{-}RelatedLeft}) \cup ((\mathsf{Related}(x_i, y) = 0) \wedge \mathsf{Err\text{-}RelatedLeft})$$

We know that $\Pr[(\mathsf{Related}(x_i, y) = 1) \wedge \mathsf{Err\text{-}RelatedLeft}] \leq \Pr[\mathsf{Err\text{-}RelatedRight}] \leq 1/\log n + \mathsf{negl}(\lambda)$. Therefore, it suffices to show that $\Pr[(\mathsf{Related}(x_i, y) = 0) \wedge \mathsf{Err\text{-}RelatedLeft}] \leq 1/\log n + \mathsf{negl}(\lambda)$.

We now consider an idealized experiment CExpt-Ideal where each PRF is replaced with a random oracle. In this experiment CExpt-Ideal, we only care about the bad event Err-RelatedLeft, and therefore we omit writing all other error messages.

Experiment CExpt-Ideal

Offline setup. For $j = 1$ to $\mathsf{lenT} := 6\sqrt{n} \cdot \log^3 n$: sample a random oracle RO and let $T_j := \mathsf{RO}$. Set $\mathsf{label}(T_j) := \perp$.

Online query for index $x \in \{0, 1, \ldots, n-1\}$.

a) Sample a new RO^* such that the associated set contains x. Append RO^* to the table T as the last entry, and mark its label $\mathsf{label}(T_{\mathsf{lenT}+1}) := \perp$.

b) Let $T_j := \mathsf{RO}_j$ be the smallest entry in the table T such that the set generated by RO_j contains x.

c) If $z := \mathsf{label}(T_j) \neq \perp$ and the set generated by RO_j contains some y such that $\mathsf{Related}(y, z) = 1$, then return Err-RelatedLeft.

d) Sample a new RO' such that the generated set contains x. Overwrite $T_j := \mathsf{RO}'$ and set $\mathsf{label}(T_j) := x$.

e) Remove the last entry from T and return Success.

Fact 7.9. *Suppose that the* PRSet *scheme satisfies pseudorandomness. Then,*

$$\Pr_{\mathsf{CExpt}}(\mathsf{Err\text{-}RelatedLeft}) \leq \Pr_{\mathsf{CExpt\text{-}Ideal}}(\mathsf{Err\text{-}RelatedLeft}) + \mathsf{negl}(\lambda)$$

where $\Pr_{\mathsf{CExpt}}(\mathsf{Err\text{-}RelatedLeft})$ *denotes the probability that the* i*-th query encounters* Err-RelatedLeft *in a random execution of* CExpt*, and* $\Pr_{\mathsf{CExpt\text{-}Ideal}}(\mathsf{Err\text{-}RelatedLeft})$ *is similarly defined but for* CExpt-Ideal *instead.*

Proof. The event Err-RelatedLeft is a polynomial-time checkable event defined over the outputs of multiple instances of PRF (or RO, respectively). Therefore, we can prove this fact through a straightforward reduction to the pseudorandomness of the PRSet scheme, through a standard hybrid argument where we replace each independent instance of PRF with an RO one by one. □

Given Fact 7.9, it suffices to prove that the probability of Err-RelatedLeft in CExpt-Ideal is upper bounded by $2/\log n$. The challenge in arguing this is that conditioned on the i-th query finding the smallest matching entry $T_j := \mathsf{RO}_j$, the distribution of RO_j is no longer uniform at random and independent of $\mathsf{label}(T_j)$. Therefore, we need a more involved probabilistic argument.

To make it easier to analyze the distribution, we can view the experiment CExpt-Ideal in an alternative way. We now imagine an experiment $\mathsf{CExpt\text{-}Ideal}^*$ that is equivalent to CExpt-Ideal but the sampling of a subset of the coins of the random oracles are deferred to the time of consumption. More specifically, in $\mathsf{CExpt\text{-}Ideal}^*$, the RO_j associated with each table entry T_j is stored in the following format — henceforth we say that a query string $y \in \{0,1\}^{\leq \log n + B}$ to a random oracle is related to $x \in \{0,1\}^{\log n}$ if x and y share a common suffix of length at least $\frac{1}{2}\log n + 1$.

- If $z := \mathsf{label}(T_j) = \perp$, RO_j is sampled when the table entry is generated or updated, and the answers to all queries are stored.
- If $z := \mathsf{label}(T_j) \neq \perp$, then for all queries not related to z, their answers are pre-sampled and stored when the table entry is updated; however, for answers to all queries related to z, we store a refined distribution Distr in the table entry characterizing all the decisions that have been made so far. Every time we need to make a decision about an element related to z, we sample the answer to this decision according to the distribution Distr, and we then refine the distribution Distr based on the newly sampled decision.

Using this as a guideline, we now rewrite CExpt-Ideal into $\mathsf{CExpt\text{-}Ideal}^*$. Since we do not care about other errors besides Err-RelatedLeft, we omit reporting some of the other types of errors in $\mathsf{CExpt\text{-}Ideal}^*$.

CExpt-Ideal*

Offline setup. For $j = 1$ to $\mathsf{lenT} := 6\sqrt{n} \cdot \log^3 n$: sample the answers to all random oracle queries, and store them in T_j; moreover, set $\mathsf{label}(T_j) := \perp$.

Online query for index $x \in \{0, 1, \ldots, n-1\}$.

1. Sample a random oracle including answers to all queries, and append the full description to the table T as the last entry; mark its label $\mathsf{label}(T_{\mathsf{lenT}+1}) := \perp$.
2. For $j' \in [1, \mathsf{lenT} + 1]$ sequentially:
   - If $y := \mathsf{label}(T_{j'}) = \perp$, check if x is in the set described by $T_{j'}$. If so, let $j := j'$ and break.
   - If x is not related to $y := \mathsf{label}(T_{j'})$, check if x is in the set described by $T_{j'}$ by looking at the pre-sampled answers for queries not related to y. If so, let $j := j'$ and break.
   - If x is related to $y := \mathsf{label}(T_{j'})$, let $\mathsf{Distr}_{j'}$ be the current description of the distribution for answers to queries related to y. Sample the binary decision b whether x is in the set associated with $T_{j'}$ based on $\mathsf{Distr}_{j'}$, and then refine the distribution $\mathsf{Distr}_{j'}$ based on the sampled result. If $b = 1$, then let $j := j'$ and break.
3. If $y := \mathsf{label}(T_j) \neq \perp$ and the set associated with T_j contains another element related to y, then return Err-RelatedLeft.
4. Partially sample a new random oracle. For all queries not related to x, sample all their answers and store them in T_j. For all queries related to x, update the distribution Distr_j to be random but subject to that x being in the set. Set $\mathsf{label}(T_j) := x$.
5. Remove the last entry of T and return Success.

In CExpt-Ideal*, conditioned on $\mathsf{Related}(x_i, y) = 0$, i.e., the i-th query finding some T_j such that $y := \mathsf{label}(T_j)$ is not related to x_i, then the description Distr_j associated with T_j must be the following: sample at random but possibly subject to the constraint that some set of elements related to y are not in the set. Note also that whether an element related to y is in the sampled set is independent of all the answers to queries not related to y. Therefore,

$$\begin{aligned}\Pr[\neg\text{Err-RelatedLeft} | (\mathsf{Related}(x_i, y) = 0) \wedge (y \neq \perp)] &\geq \Pr_{S \xleftarrow{\$} \mathcal{D}_n^{+y}}[\nexists z \in S \text{ s.t. } \mathsf{Related}(z, y) = 1] \\ &\geq 1 - 1/\log n \qquad \text{(due to Lemma 5.2)}\end{aligned}$$

Therefore, we have that

$$\begin{aligned}\Pr[(\mathsf{Related}(x_i, y) = 0) \wedge \text{Err-RelatedLeft}] &\leq \Pr[\text{Err-RelatedLeft} | (\mathsf{Related}(x_i, y) = 0) \wedge (y \neq \perp)] \\ &\leq 1 - \Pr[\neg\text{Err-RelatedLeft} | (\mathsf{Related}(x_i, y) = 0) \wedge (y \neq \perp)] \\ &\leq 1/\log n\end{aligned}$$

Summarizing the above, we have that $\Pr[\text{Err-RelatedLeft}] \leq 2/\log n$ in CExpt-Ideal*. □

Next, we shall bound the probability of Err-RemoveFailLeft.

Lemma 7.10. *Suppose that* PRSet *satisfies (the second property in) security w.r.t. puncturing. Then,* $\Pr[\text{Err-RemoveFailLeft}] \leq 1/(\sqrt{n}\log^2 n) + \mathsf{negl}(\lambda)$ *in* CExpt.

Proof. Let us first rewrite CExpt by 1) removing all instructions not relevant to the bad event $\mathsf{Err\text{-}RemoveFailLeft}$; and 2) computing whether the punctured set contains the point being punctured when the table entry is being updated, not when being consumed. In this way, we obtain CExpt' as described below:

Experiment CExpt'

Offline setup. For $j = 1$ to $\mathsf{lenT} := 6\sqrt{n} \cdot \log^3 n$: sample $(\mathsf{sk}_j, \_) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$; let $T_j := \mathsf{sk}_j$, and set $\mathsf{label}(T_j) := \mathsf{good}$.

Online query for index $x \in \{0, 1, \ldots, n-1\}$**.**

(a) Sample $(\mathsf{sk}^*, \_) \xleftarrow{\$} \mathsf{PRSet}(1^\lambda, n)$ subject to $x \in \mathsf{PRSet.Set}(\mathsf{sk}^*)$. Append sk^* to the table T as the last entry, and mark its label $\mathsf{label}(T_{\mathsf{lenT}+1}) := \mathsf{good}$.

(b) Let $T_j := \mathsf{sk}_j$ be the smallest entry in the table T such that $x \in \mathbf{Set}(\mathsf{sk}_j)$.

(c) If $z := \mathsf{label}(T_j) = \mathsf{bad}$ and then return $\mathsf{Err\text{-}RemoveFailLeft}$.

(d) Sample a new $(\mathsf{sk}', \mathsf{msk}') \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$ subject to $x \in \mathsf{PRSet.Set}(\mathsf{sk}')$. Call $\mathsf{sk}'' \leftarrow \mathsf{PRSet.Puncture}(\mathsf{msk}', x)$. Overwrite $T_j := \mathsf{sk}''$. Moreover, if $x \in \mathbf{Set}(\mathsf{sk}'')$, set $\mathsf{label}(T_j) := \mathsf{bad}$; else set $\mathsf{label}(T_j) := \mathsf{good}$.

(e) Remove the last entry from T, and return $\mathsf{Success}$.

It suffices to prove that in CExpt', $\Pr[\mathsf{Err\text{-}RemoveFailLeft}] \leq 1/(\sqrt{n} \cdot \log^2 n) + \mathsf{negl}(\lambda)$. We now focus on analyzing CExpt'.

We consider the following hybrid experiment Hyb. Hyb is defined almost the same way as CExpt', except the following modification: Step (d) is replaced with the following: sample $(\mathsf{sk}', \_) \leftarrow \mathsf{PRSet.Gen}(1^\lambda, n)$ subject to $x \in \mathbf{Set}(\mathsf{sk}')$, and let $b \leftarrow \mathsf{Bernoulli}(2^{-(\frac{1}{2}\log n + B)})$. If $b = 1$, mark $\mathsf{label}(T_j) = \mathsf{bad}$; else, mark $\mathsf{label}(T_j) = \mathsf{good}$.

Due to the "security w.r.t. puncturing" property of PRSet, $\Pr_{\mathsf{CExpt}'}[\mathsf{Err\text{-}RemoveFailLeft}] \leq \Pr_{\mathsf{Hyb}}[\mathsf{Err\text{-}RemoveFailLeft}] + \mathsf{negl}(\lambda)$ for an arbitrary query i. In Hyb, clearly, $\Pr[\mathsf{Err\text{-}RemoveFailLeft}] \leq 2^{-(\frac{1}{2}\log n + B)} \leq 1/(\sqrt{n} \cdot \log^2 n)$.

□

8 Final Construction: Multi-Copy Scheme

Multi-Copy Scheme PIR^k

Run k parallel copies of the single-copy scheme in Section 7 with the following modification: recall that in the single-copy scheme, Client obtains the true answer β from an oracle; in our multi-copy scheme, Client sets $\beta := \mathsf{Maj}(\beta'_1, \ldots, \beta'_k)$ where for $i \in [k]$, β'_i denotes the candidate answer β' of the i-th copy, and $\mathsf{Maj}(\cdot)$ is the majority function. β is also output as the answer to the query.

Fact 8.1 (Correctness of the multi-copy scheme)**.** *For any* $\mathsf{DB} \in \{0,1\}^n$*, for any* Q *that is polynomially bounded in* λ*, for any sequence of queries* $x_1, x_2, \ldots, x_Q \in \{0, 1, \ldots, n-1\}$*, for any* $i \in [Q]$*, with at least* $1 - \frac{1}{2^{\Theta(k)}}$ *probability, the majority-vote outcome* β *for the* i*-th query in the above multi-copy scheme is correct.*

Proof. Due to Theorem 7.7, each individual copy gives a correct answer with probability at least $2/3$. Therefore, the expected number of correct answers among the k copies is $2k/3$. Since each copy is correct with independent probability, due to the Chernoff bound, the probability that at least $k/2$ copies return incorrect answers is upper bounded by $\exp(-\Theta(k))$. □

Theorem 8.2 (Multi-copy PIR scheme)**.** *Suppose that the underlying single-copy scheme satisfies privacy (i.e., Theorems 7.2 and 7.3) and occasional correctness (i.e., Theoerm 7.7). Then, the above multi-copy PIR scheme where $k = \omega(\log \lambda)$ satisfies correctness and privacy as defined in Section 6.*

Proof. Correctness follows from Fact 8.1 which in turn relies on Theorem 7.7. Further, Thereom 7.7 holds as long as the β value in the single-copy scheme is indeed correct in all steps. Privacy follows directly from the privacy of each single-copy scheme, namely, Theorem 7.2 and Theorem 7.3. □

This immediately gives rise to the following corollary. Henceforth, let $\chi(\lambda)$ be an upper bound on the underlying puncturable PRF's (punctured) secret key length and evaluation time; $\chi(\lambda)$ is upper bounded by a polynomial in the security parameter λ, and is related to the strength of the underlying cryptographic assumptions. For example, using the private puncturable PRF schemes of either Boneh, Kim, and Montgomery [BKM17] or Canetti and Chen [CC17], $\chi(\lambda) = \mathsf{poly}\log(\lambda)$ if we assumed sub-exponential security of LWE; and $\chi(\lambda) = \mathsf{poly}(\lambda)$ if we assumed polynomial security of LWE.

Corollary 8.3 (2-server preprocessing PIR)**.** *Assume the hardness of Learning With Errors (LWE). Then, there exists a two-server offline/online PIR scheme that satisfies privacy and correctness, and the following performance bounds where $\chi(\lambda)$ denotes a security parameter related to the strength of the underlying cryptographic primitive as explained above:*

- *the offline server runtime is $\chi(\lambda) \cdot O(n) \cdot \mathsf{poly}\log(n, \lambda)$; the offline client runtime and bandwidth is $\chi(\lambda) \cdot O(\sqrt{n}) \cdot \mathsf{poly}\log(n, \lambda)$;*
- *the online server and client runtime per query is $\chi(\lambda) \cdot O(\sqrt{n}) \cdot \mathsf{poly}\log(n, \lambda)$; and the online communication per query is $\chi(\lambda) \cdot \mathsf{poly}\log(n, \lambda)$;*
- *each server needs to store only the original database DB and no extra information; and the client stores $\chi(\lambda) \cdot O(\sqrt{n}) \cdot \mathsf{poly}\log(n, \lambda)$ state.*

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, SP 2018*, pages 962–979, 2018.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 1292–1303, 2016.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In *CRYPTO*, pages 55–73, 2000.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In *TCC*, volume 10678, pages 662–693, 2017.

[BKM17] Dan Boneh, Sam Kim, and Hart William Montgomery. Private puncturable PRFs from standard lattice assumptions. In *EUROCRYPT*, pages 415–445, 2017.

[BKW17] Dan Boneh, Sam Kim, and David J. Wu. Constrained keys for invertible pseudorandom functions. In *TCC*, volume 10677, pages 237–263. Springer, 2017.

[BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In *International Conference on Practice and Theory in Public-Key Cryptography (PKC)*, 2017.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained prfs (and more) from lwe. In *TCC*, pages 264–302, 2017.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In *EUROCRYPT*, pages 446–476, 2017.

[CG97a] Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 304–313, New York, NY, USA, 1997. ACM.

[CG97b] Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*, pages 304–313, 1997.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 41–50, 1995.

[Cha04] Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In *Information Security and Privacy: 9th Australasian Conference (ACISP)*, pages 50–61, 2004.

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In *TCC*, volume 10678, pages 694–726, 2017.

[CK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *EUROCRYPT 2020*, volume 12105, pages 44–75, 2020.

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

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, pages 402–414, 1999.

[DCIO98] Giovanni Di-Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Universal service-providers for database private information retrieval. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, page 91–100, 1998.

[DG16] Zeev Dvir and Sivakanth Gopi. 2-server pir with subpolynomial communication. *J. ACM*, 63(4), 2016.

[DHS14] Daniel Demmler, Amir Herzberg, and Thomas Schneider. Raid-pir: Practical multi-server pir. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security (CCSW)*, page 45–56, 2014.

[DvDF+16] Srinivas Devadas, Marten van Dijk, Christopher W. Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion ORAM: A constant bandwidth blowup oblivious RAM. In Eyal Kushilevitz and Tal Malkin, editors, *TCC*, volume 9563, pages 145–174, 2016.

[Gas04] William I. Gasarch. A survey on private information retrieval (column: Computational complexity). *Bulletin of the EATCS*, 82:72–107, 2004.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4), August 1986.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled ram revisited. In *EUROCRYPT*, pages 405–422, 2014.

[GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *CRYPTO*, volume 9816, pages 563–592, 2016.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 1996.

[Gol87] O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *STOC*, 1987.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 803–815, 2005.

[HH17] Syed Mahbub Hafiz and Ryan Henry. Querying for queries: Indexes of queries for efficient and expressive IT-PIR. In *ACM CCS*, pages 1361–1373, 2017.

[HOWW19] Ariel Hamlin, Rafail Ostrovsky, Mor Weiss, and Daniel Wichs. Private anonymous data access. In *EUROCRYPT*, volume 11477, pages 244–273, 2019.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing (STOC)*, page 262–271, 2004.

[IKOS06] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *FOCS*, pages 239–248, 2006.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *FOCS*, 1997.

[LG15] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In *Financial Cryptography and Data Security (FC)*, volume 8975, pages 168–186, 2015.

[Lip10] Helger Lipmaa. First CPIR protocol with data-dependent computation. In *Proceedings of the 12th international conference on Information security and cryptology*, ICISC'09, pages 193–210, Berlin, Heidelberg, 2010. Springer-Verlag.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble ram programs. In *EUROCRYPT*, 2013.

[MR14] Ben Morris and Phillip Rogaway. Sometimes-recurse shuffle - almost-random permutations in logarithmic expected time. In *Eurocrypt*, volume 8441, pages 311–326. Springer, 2014.

[OS07] Rafail Ostrovsky and William E. Skeith, III. A survey of single-database private information retrieval: techniques and applications. In *PKC*, pages 393–411, 2007.

[PPY18] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, page 1002–1019, 2018.

[PR93] P. Pudlák and V. Rödl. Modified ranks of tensors and the size of circuits. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 523–531, 1993.

[RY13] Thomas Ristenpart and Scott Yilek. The mix-and-cut shuffle: Small-domain encryption secure against N queries. In *CRYPTO*, volume 8042, pages 392–409, 2013.

[SCSL11] Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $O((\log N)^3)$ worst-case cost. In *ASIACRYPT*, 2011.

[SS12] Emil Stefanov and Elaine Shi. Fastprp: Fast pseudo-random permutations for small domains. *IACR Cryptol. ePrint Arch.*, 2012:254, 2012.

[TDG16] Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-cost ϵ-private information retrieval. *Proc. Priv. Enhancing Technol.*, 2016(4):184–201, 2016.