Near-Optimal Private Information Retrieval with Preprocessing

Arthur Lazzaretti and Charalampos Papamanthou

Yale University
{arthur.lazzaretti,charalampos.papamanthou}@yale.edu

Abstract. In Private Information Retrieval (PIR), a client wishes to access an index i from a public n-bit database without revealing any information about i. Recently, a series of works starting with the seminal paper of Corrigan-Gibbs and Kogan (EUROCRYPT 2020) considered PIR with *client preprocessing* and *no additional server storage*. In this setting, we now have protocols that achieve $\widetilde{O}(\sqrt{n})$ (amortized) server time and $\widetilde{O}(1)$ (amortized) bandwidth in the two-server model (Shi et al., CRYPTO 2021) as well as $\widetilde{O}(\sqrt{n})$ server time and $\widetilde{O}(\sqrt{n})$ bandwidth in the single-server model (Corrigan-Gibbs et al., EUROCRYPT 2022). Given existing lower bounds, a single-server PIR scheme with $\widetilde{O}(\sqrt{n})$ (amortized) server time and $\widetilde{O}(1)$ (amortized) bandwidth is still feasible, however, to date, no known protocol achieves such complexities. In this paper we fill this gap by constructing the first single-server PIR scheme with $\widetilde{O}(\sqrt{n})$ (amortized) server time and $\widetilde{O}(1)$ (amortized) bandwidth. Our scheme achieves near-optimal (optimal up to polylogarithmic factors) asymptotics in every relevant dimension. Central to our approach is a new cryptographic primitive that we call an *adaptable pseudorandom set*: With an adaptable pseudorandom set, one can represent a large pseudorandom set with a succinct fixed-size key k, and can both add to and remove from the set a constant number of elements by manipulating the key k, while maintaining its concise description as well as its pseudorandomness (under a certain security definition).

1 Introduction

In private information retrieval (PIR), a server holds a public database DB represented as an n-bit string and a client wishes to retrieve $\mathsf{DB}[i]$ without revealing i to the server. PIR has many applications in various systems with advanced privacy requirements [2,3, 28, 31, 37] and comprises a foundational computer science and cryptography problem, with connections to primitives such as oblivious transfer [19] and locally-decodable codes [29,39], among others. PIR can be naively realized by downloading the whole DB for each query, which is prohibitive for large databases. PIR is classically considered within the two-server model [11, 12, 14], where DB is replicated on two, non-colluding servers. *For the rest of the paper we use* **1PIR** *to refer to single-server PIR [32] and* **2PIR** *to refer to two-server PIR.* Clearly, 1PIR is much more challenging than 2PIR, but also more useful; it is hard to ensure two servers do not collude and remain both synchronized and available in practice [6, 35].

Sublinear time 2PIR. Preliminary PIR works [4, 13, 20–22, 25, 30, 32–34, 38] featured linear server time and sublinear bandwidth. To reduce server time, several works [1, 5, 17, 18, 23, 28] proposed *preprocessing PIR*. These approaches require a prohibitive amount of server storage due to large server-side data structures. Recently a new type of preprocessing PIR with *offline client-side preprocessing* was proposed by Corrigan-Gibbs and Kogan [16]. Introduced as 2PIR, their scheme has sublinear server time and *no additional server storage* — the preprocessing phase outputs just a few bits to be stored at the client, which is modeled as stateful. A simplified, stripped-down[1] version of their protocol, involving three parties, **client**, **server**$_1$ and **server**$_2$, is given below.

- *Offline phase.* **client** sends $S_1, \ldots, S_{\sqrt{n}}$ to **server**$_1$. Each S_i is independent and contains $\sqrt{n}$ elements sampled uniformly from $\{0, \ldots, n-1\}$ without replacement. **server**$_1$ returns database parities $p_1, \ldots, p_{\sqrt{n}}$, where $p_i = \oplus_{j \in S_i} \mathsf{DB}[j]$. These database parities, along with the respective index sets, are then stored by **client** locally.
- *Online phase (query to index i).* In Step 1, **client** finds a local set S_j that contains i and sends $S'_j = S_j \setminus \{i\}$ to **server**$_2$. In Step 2, **server**$_2$ returns parity p'_j of S'_j, and **client** computes $\mathsf{DB}[i] = p_j \oplus p'_j$. In Step 3, **client** generates a fresh random set S^*_j that contains i, sends $S^*_j \setminus \{i\}$ to **server**$_1$, gets back its parity p^*_j, and replaces (S_j, p_j) with $(S^*_j, p^*_j \oplus \mathsf{DB}[i])$. (We note that this last step is crucially needed to maintain the distribution of the sets at the client side and ensure security of future queries.)

The complexities of the above protocol are linear (such as client storage and bandwidth), but Corrigan-Gibbs and Kogan [16] achieved $\widetilde{O}(\sqrt{n})$ time and communication complexities by introducing the notion of *pseudorandom sets*: Instead of sending the sets in plaintext, the client sends a Pseudorandom Permutation (PRP) key so that the server can regenerate the sets as well as check membership efficiently. However, the first step of the online phase above requires removing element i from the set S_j. This cannot be done efficiently with a PRP key, so prior work sends $S_j \setminus \{i\}$ in plaintext, incurring $O(\sqrt{n} \log n)$ online bandwidth. In a followup work, Shi et al. [36] addressed this issue. They use no PRPs and construct their sets via *privately-puncturable pseudorandom functions* [7,10]. Their primitive allows element removal without key expansion in the online phase, thus keeping a short set description, yielding $\widetilde{O}(1)$ bandwidth.

Compiling 2PIR into 1PIR. The original protocol by Corrigan-Gibbs and Kogan [16], their follow-up work [31], as well as Shi et al.'s polylog bandwidth protocol [36], are all 2PIR protocols. Corrigan-Gibbs et al. [15] showed how to port the 2PIR protocols by Corrigan-Gibbs and Kogan [16, 31] into a 1PIR scheme with the same (amortized[2]) $\widetilde{O}(\sqrt{n})$ complexities. Their main technique, is to transform their initial 2PIR scheme [15] into another 2PIR scheme that *avoids communication with* **server**$_1$ *in the*

[1] In particular, in Step 1 of the actual protocol's online phase, the client sends $S_j \setminus \{i\}$ with probability $1 - 1/\sqrt{n}$ and $S_j \setminus \{r\}$, for a random element r, with probability $1/\sqrt{n}$, to ensure no information is leaked about i. Also, $\omega(\log \lambda)$ parallel executions are required to guarantee overwhelming correctness in λ, to account for puncturing 'fails' and when a set S_j that contains i cannot be found.

[2] Amortization is over $\sqrt{n}$ queries.

Table 1. Comparison with related work. Server time and bandwidth are amortized (indicated with a *). All schemes presented have $\widetilde{O}(\sqrt{n})$ client time, $\widetilde{O}(\sqrt{n})$ client space and no additional server space. The amortization kicks in after $\sqrt{n}$ queries.

scheme	model	server time*	bandwidth*	assumption
[15]	1PIR	$\widetilde{O}(\sqrt{n})$	$\widetilde{O}(\sqrt{n})$	LWE
[36]	2PIR	$\widetilde{O}(\sqrt{n})$	$\widetilde{O}(1)$	LWE
Theorem 52	1PIR	$\widetilde{O}(\sqrt{n})$	$\widetilde{O}(1)$	LWE

online phase. We call such a 2PIR protocol 2PIR+. Then, they use fully-homomorhpic encryption (FHE) [24] to execute both offline and online phases on the same server, yielding 1PIR. To build the crucial 2PIR+ protocol, they make two simple modifications of the high-level protocol presented before: (i) In the offline phase, instead of preprocessing $\sqrt{n}$ sets, they preprocess $2\sqrt{n}$ sets, where $\sqrt{n}$ is the number of queries they wish to support; (ii) In the final step of the online phase, instead of picking a fresh random set S_j^* and then communicating with $\mathbf{server}_1$, they use a preprocessed set S_h from above, *avoiding communication with* $\mathbf{server}_1$ *in the online phase*. Crucially, S_h must then be updated to contain i, so that the primary sets maintain the same distribution after each query. After $\sqrt{n}$ queries there are no more preprocessed sets left and the offline phase is run again, maintaining the same amortized complexity.[3]

Based on the above, it seems that a natural approach to construct a sublinear-time, polylog-bandwidth 1PIR scheme (which is the central contribution of this paper) would be to apply the same trick of preprocessing an additional $\sqrt{n}$ random sets to the Shi et al. protocol [36]. But this strategy runs into a fundamental issue: We would have to ensure that, in Step 3 of the online phase, when we use one of the preprocessed sets, S_h, to replace the set that was just consumed to answer query i, the set key corresponding to S_h *would have to be updated to contain* i. However, this is not supported in the current construction of pseudorandom sets by Shi et al. [36]—one can only remove elements, but not add. Our work capitalizes on this observation.

Technical highlight: Adaptable pseudorandom sets. A substantial part of our contribution is to define and construct an *adaptable pseudorandom set* supporting *both* element removal and addition. In fact, our technique can support addition and removal of a logarithmic number of elements. At a high level, our primitive can be used as follows. Key generation outputs a succinct key sk representing the set. Along with algorithms for enumeration of sk and membership checking in sk, we define algorithms for removing an element x from the set defined by sk and adding an element x into the set defined by sk, both of which output the updated set's new key sk'. We believe that this primitive can also be of independent interest outside of PIR.

Our construction of adaptable PRSets is simple. First, we show how previous puncturable pseudorandomsets can be modified to support a single addition (instead of a single removal). Then, we show that given both capabilities, one can compose pseudo-

[3] We pick $\sqrt{n}$ concretely for exposition. Looking ahead, our scheme achieves a same smooth tradeoff where by preprocessing $O(Q)$ sets achieves $O(n/Q)$ amortized online time.

random set keys to support any number of additions and removals. For the usecase of PIR, it is sufficient to support exactly one removal and one addition, but the technique can be extended further.

Our final 2PIR+ and 1PIR protocols. Armed with adaptable pseudorandom sets, a high-level description of our new 2PIR+ scheme is as follows. Below, APRS denotes "adaptable pseudorandom set".

- *Offline phase.* **client** sends $\sqrt{n}+Q$ APRS keys $sk_1, \ldots, sk_{\sqrt{n}+Q}$ to $\mathbf{server}_1$ and $\mathbf{server}_1$ returns database parities $p_1, \ldots, p_{\sqrt{n}+Q}$ where $p_i = \oplus_{j \in sk_i} \mathsf{DB}[j]$. The database parities are then stored locally by **client**, together with the respective APRS keys.
- *Online (query to index i).* First, **client** finds APRS key sk_j that contains i, **removes** i from sk_j and sends sk'_j to $\mathbf{server}_2$. Then $\mathbf{server}_2$ returns parity p'_j of sk'_j, and **client** computes $\mathsf{DB}[i] = p_j \oplus p'_j$. Finally, **client adds** i into key sk_h (for some $h > \sqrt{n}$) and replaces (sk_j, p_j) with $(sk_h, p_h \oplus \mathsf{DB}[i])$.

The above 2PIR+ protocol requires more work to ensure a small probability of failure and that the server's view is uniform. Also, again, we can convert the above 2PIR+ protocol to 1PIR with sublinear complexities, using FHE [15]. Note that using FHE naively for 1PIR would incur $\widetilde{O}(n)$ server time—thus combining FHE with our above 2PIR+ protocol yields a much better (sublinear) FHE-based 1PIR instantiation.

Our result and comparison with related work. As we discussed, if we require the server time to be sublinear (with no additional storage), the most bandwidth-efficient 2PIR protocol is the one by Shi et al. [36]. However, the most efficient 1PIR construction, by Corrigan-Gibbs et al. [15], incurs bandwidth on the order of $O(\sqrt{n}\log n)$.

In this paper, we fill this gap. Our result (Theorem 52) provides the first 1PIR protocol with *sublinear amortized server time and polylogarithmic amortized bandwidth*.

We note that our scheme is optimal up to polylogarithmic factors in every relevant dimension, given known lower bounds for client-dependent preprocessing PIR where the server stores only the database [5, 15, 16]. For a comparison with prior sublinear-server-time-no-additional-server-storage schemes, see Table 1.

Concurrent work. We note independently and concurrently, the notion of 1PIR with polylogarithmic bandwidth and sublinear server time was studied by Zhou et al. [40]. Their work requires use of a privately *programmable* PRF, and the sets constructed do not enjoy the same strong security properties as our adaptable pseudorandom sets. Specifically, our adaptable sets are defined more generally. One can pick L = $O(\log(N))$ (for sets of size N) additions or removals to support when generating the set key, and the set will support any number between 0 and L of adaptive additions/removals, maintaining a concise description, and with each intermediate key satisfying our security definitions. Our adaptable PRSets could therefore have more applications due to their higher flexibility. With respect to the final PIR scheme, the asymptotics achieved in their scheme are the same as the asymptotics achieved here in every dimension (what we define as near-optimality).

Notation. We use the abbreviation PPT to refer to probabilistic polynomial time. Unless otherwise noted, we define a negligible function $\mathtt{negl}(\cdot)$ to be a function such that for every polynomial $p(\cdot)$, $\mathtt{negl}(\cdot)$ is less than $1/p(\cdot)$. We fix $\lambda \in \mathbb{N}$ to be a security

parameter. We will also use the notation 1^z or 0^z to represent 1 or 0 repeated z times. For any vector or bitstring V, we index V using the notation $V[i]$ to represent the i-th element or i-th bit of V, indexed from 0. We will also use the notation $V[i:]$ to denote V from the i-th index onwards. We use $x||y$ to denote the concatenation of bitsring x and bitstring y. We use $S \sim D$ to denote that S is "sampled from distribution" D. We use the notation $[x, y]$ to represent the set $\{x, x+1, \ldots, y-1\}$. Finally, we use $\widetilde{O}(\cdot)$ to denote the big-O notation that ignores polylogarithmic terms and any polynomial terms in the security parameter λ.

2 Background: PIR, Puncturable Functions and Puncturable Sets

We now introduce definitions for 2PIR. We consider 2PIR protocols where only one server (the second one) participates in the online phase. We refer to these protocols as 2PIR+. We also formally introduce privately-puncturable PRFs [7] and privately-puncturable pseudorandom sets [16, 36], both crucial for our work. Moving forward, "PRF" stands for "pseudorandom function" and "PRS" stands for "pseudorandom set".

Definition 21 (2PIR+ scheme)**.** *A 2PIR+ scheme consists of three stateful algorithms (***server**$_1$**,** **server**$_2$**,** **client***) with the following interactions.*

- ***Offline****:* **server**$_1$ *and* **server**$_2$ *receive the security parameter* 1^λ *and an* n*-bit database* DB*.* **client** *receives* 1^λ*.* **client** *sends one message to* **server**$_1$ *and* **server**$_1$ *replies with one message.*
- ***Online****: For any query* $x \in \{0, \ldots, n-1\}$*,* **client** *sends one message to* **server**$_2$ *and* **server**$_2$ *responds with one message. In the end,* **client** *outputs a bit* b*.*

Definition 22 (2PIR+ correctness)**.** *A 2PIR+ scheme is correct if its honest execution, with any database* $\mathsf{DB} \in \{0,1\}^n$ *and any polynomial-sized sequence of queries* $x_1, \ldots, x_Q$*, returns* $\mathsf{DB}[x_1], \ldots, \mathsf{DB}[x_Q]$ *with probability* $1 - \mathtt{negl}(\lambda)$*.*

Definition 23 (2PIR+ privacy)**.** *A 2PIR+ scheme (***server**$_1$**,** **server**$_2$**,** **client***) is private if there exists a PPT simulator* $\mathtt{Sim}$*, such that for any algorithm* $serv_1$*, no PPT adversary* $\mathcal{A}$ *can distinguish the experiments below with non-negligible probability.*

- ***Expt***$_0$*:* **client** *interacts with* $\mathcal{A}$ *who acts as* **server**$_2$ *and* ***server***$_1^*$ *who acts as the* **server**$_1$*. At every step* t*,* $\mathcal{A}$ *chooses the query index* x_t*, and* **client** *is invoked with input* x_t *as its query and outputs its query.*
- ***Expt***$_1$*:* $\mathtt{Sim}$ *interacts with* $\mathcal{A}$ *who acts as* **server**$_2$ *and* ***server***$_1^*$ *who acts as the* **server**$_1$*. At every step* t*,* $\mathcal{A}$ *chooses the query index* x_t*, and* $\mathtt{Sim}$ *is invoked with no knowledge of* x_t *and outputs a query.*

We note that in the above definition our adversary $\mathcal{A}$ can deviate arbitrarily from the protocol. Intuitively the privacy definition implies that queries made to **server**$_2$ will appear random to **server**$_2$, assuming servers do not collude (as is the case in our model). Also, note that the above definition only captures privacy for **server**$_2$ since by Definition 21, **server**$_1$ interacts with **client** *before* the query indices are picked.

Privately-puncturable PRFs. A puncturable PRF is a PRF F whose key k can be punctured at some point x in the domain of the PRF, such that the output punctured key

k_x reveals nothing about $F_k(x)$ [27]. A privately-puncturable PRF is a puncturable PRF where the punctured key k_x also reveals *no* information about the punctured point x (by re-randomizing the output $F_k(x)$). Privately-puncturable PRFs can be constructed from standard LWE (learning with errors assumption) [7, 8, 10] and can be implemented to allow puncturing on m points at once [7]. We now give the formal definition.

Definition 24 (Privately-puncturable PRF [7])**.** *A privately-puncturable PRF with domain $\{0,1\}^*$ and range $\{0,1\}$ has four algorithms: (i) $\mathsf{Gen}(1^\lambda, L, m) \to sk$: Outputs secret key sk, given security parameter λ, input length L and number of points to be punctured m; (ii) $\mathsf{Eval}(sk, x) \to b$: Outputs the evaluation bit $b \in \{0,1\}$, given sk and input x; (iii) $\mathsf{Puncture}(sk, P) \to sk_P$: Outputs punctured key sk_P, given sk and set P of m points for puncturing; (iv) $\mathsf{PEval}(sk_P, x) \to b$: Outputs the evaluation bit $b \in \{0,1\}$, given sk_P and x.*

There are three properties we require from a privately-puncturable PRF: First, *functionality preservation*, meaning that $\mathsf{PEval}(sk_P, x)$ equals $\mathsf{Eval}(sk, x)$ for all $x \notin P$. Second, *pseudorandomness*, meaning that the values $\mathsf{Eval}(sk, x)$ at $x \in P$, appear pseudorandom to the adversary that has access to sk_P *and* oracle access to $\mathsf{Eval}(sk, \cdot)$ (as long as the adversary cannot query $\mathsf{Eval}(sk, x)$ for $x \in P$, in which case it is trivial to distinguish). Third, *privacy with respect to puncturing*, meaning that the punctured key sk_P does not reveal anything about the set of points that was punctured. Formal definitions are given in [7] (For convenience, we also include them in our auxiliary material—see Definitions E2, E1, E3.)

It is important to note here that we will be using a privately-puncturable PRF with a *randomized* puncturing algorithm. Although initial constructions were deterministic [7], Canetti and Chen [10] show how to support randomized puncturing without extra assumptions and negligible extra cost. Any of the constructions can be extended in the manner shown in [10] to achieve a randomized puncturing. The randomization will be important since our add functionality uses rejection sampling.

Privately-puncturable PRSs. A privately-puncturable PRS is a set that contains elements drawn from a given distribution $\mathbb{D}_n$. (We define a $\mathbb{D}_n$ to be used in this work in Section 3.) The set can be represented succinctly with a key sk. Informally, one can "puncture" an element x, producing a new key that represents a set without x. Privately-puncturable PRSs were first introduced by Corrigan-Gibbs and Kogan [16] and were further optimized by Shi et al. [36]. The formal definition is as follows.

Definition 25 (Privately-puncturable PRS [16, 36])**.** *A privately-puncturable PRS has four algorithms: (i) $\mathsf{Gen}(1^\lambda, n) \to (msk, sk)$: Outputs a set key sk and a master key msk, given security parameter λ and the set domain $\{0, \ldots, n-1\}$; (ii) $\mathsf{EnumSet}(sk) \to S$: Outputs set S given sk; (iii) $\mathsf{InSet}(sk, x) \to b$: Outputs a bit b denoting whether $x \in \mathsf{EnumSet}(sk)$; (iv) $\mathsf{Resample}(msk, x) \to sk_x$: Outputs a secret key sk_x for a set generated by sk, with x's membership resampled.*[4]

We require three properties from a privately-puncturable PRS: First, *pseudorandomness with respect to a distribution $\mathbb{D}_n$*, meaning that $\mathsf{Gen}(1^\lambda, n)$ generates a key that

[4] Previously this was called "puncture". We rename it to "resample" for ease of understanding and consistency with our work.

represents a set whose distribution is indistinguishable from $\mathbb{D}_n$. Second, *functionality preservation with respect to resampling*, informally meaning that the set resulting from resampling should be a subset of the original set. This means we can only resample elements already in the set. Third, *security in resampling*, states that for any (msk, sk) output by $\mathsf{Gen}(1^\lambda, n)$, sk is computationally indistinguishable from a key sk'_x where (msk', sk') is a key output by calling $\mathsf{Gen}(1^\lambda, n)$ until $\mathsf{InSet}(sk', x) \to 1$ and sk'_x is the output of $\mathsf{Resample}(msk', x)$. Formal definitions can be found in [16, 36]. (See Definitions A3, A1, A2.)

Privately-puncturable PRSs from privately-puncturable PRFs. Shi et al. [36] constructed a privately-puncturable PRS from a privately-puncturable PRF. Let F be a privately-puncturable PRF and let $x \in \{0,1\}^{\log n}$ be an element of the set domain. We provide the intuition behind the construction. Consider that we require both concise description and fast membership testing. One first approach to constructing a PRS could be to define $x \in S$ to be $F.\mathsf{Eval}(sk, x)$ equals 1. Resampling x would then be equivalent to puncturing F's key at point x. Given $x \in \{0,1\}^{\log n}$, this approach creates sets proportional to the size of $n/2$ in expectation, which is undesirable for our application; we want sets of size approximately $\sqrt{n}$. To deal with this problem, one can add additional constraints with respect to suffixes of x. In other words, define $x \in S$ iff $F.\mathsf{Eval}(sk, x[i:])$ equals 1, for all $i = [0, \log n/2]$. Recall $x[i:]$ denotes the suffix of bitstring x starting at position i. Puncturing in this case would require puncturing at $\log n/2$ points. While this approach generates sets of expected size $\sqrt{n}$, it introduces too much dependency between elements in the set: Elements with shared suffixes are very likely to be together in the set. To deal with this, Shi et al. [36] changed the construction as follows. Let B be an integer greater than 0. Then, let $z = 0^B||x$. We say that $x \in S$ iff

$$F.\mathsf{Eval}(sk, z[i:]) = 1, \text{for all } i = [0, \log n/2 + B]\,.$$

For clarity we provide a small example here. Suppose $n = 16$ and that we want to check the membership of element 7 for set S. First, we represent 7 with $\log 16 = 4$ bits, $7_2 = 0111$. Next, we append $B = 4$ zeros to the front of the bitstring, so that we have the string 00000111. Now, we say that $7 \in S$ iff

$$F.\mathsf{Eval}(sk, 00000111) = 1 \wedge F.\mathsf{Eval}(sk, 0000111) = 1 \wedge F.\mathsf{Eval}(sk, 000111) = 1$$
$$\wedge\, F.\mathsf{Eval}(sk, 00111) = 1 \wedge F.\mathsf{Eval}(sk, 0111) = 1 \wedge F.\mathsf{Eval}(sk, 111) = 1\,.$$

Note that adding these B extra checks decreases dependency of set membership between elements proportional to 2^B, since it adds bits unique to each element. As a tradeoff, it decreases the size of the set proportional to 2^B. By picking $B = \lceil 2 \log \log n \rceil$, we maintain the set size to be $\sqrt{n}/\log^2 n$ while having small dependency between elements—which can be addressed. We give an overview of our remaining algorithms:

Set enumeration. Let $m = \log n/2 + B$. Naively, set enumeration would take linear time, since membership for each $x \in \{0, \ldots, n-1\}$ must be checked. Shi et al. [36] observed that due to the dependency introduced, the set can be enumerated in expected time $\widetilde{O}(\sqrt{n})$.

Resampling. To resample an element x from the set S, we puncture the PRF key at the $m = \log n/2 + 2\log\log n$ points that determine x's membership by running

$$sk_x \leftarrow F.\mathsf{Puncture}(sk, \{z[i:]\}_{i=[0,m]}) .$$

By the pseudorandomness of F, this will resample x's membership in S and x will not be in the set defined by sk_x with probability $1 - 1/2^m = 1 - 1/\sqrt{n}\log^2 n$. Clearly, we do not remove elements from the set with overwhelming probability. Aside from that, there is still dependency among elements, and puncturing x may also remove other elements in S with some small probability. Shi et al. [36] resolve this by bounding these probabilities to less than $1/2$ and running λ parallel executions of the protocol and taking a majority. Looking ahead, we will require this too.

Key generation. By Definition 25, key generation for a privately-puncturable PRS outputs two keys, key sk that represents the initial set and key msk that is used for puncturing. To output msk, we simply call $F.\mathsf{Gen}(1^\lambda, L, m)$. To output sk, we pick a set P of m "useless" strings of $L = \log n + B$ bits that start with the 1 bit and output a second key $sk \leftarrow F.\mathsf{Puncture}(msk, P)$. The reason for that is to ensure that resampled keys are indisinguishable from freshly sampled keys as required by the "security in resampling" property. Therefore we artificially puncture msk in a way that does not affect the set of elements represented by it, yet we change its format to be the same as a set key resampled at a given point.

Efficiency and security. To summarize, the scheme described above by Shi et al. [36] has the following complexities: Algorithms Gen, InSet and $\mathsf{Resample}$ run in $\widetilde{O}(1)$ time. All keys have $\widetilde{O}(1)$ size. Algorithm $\mathsf{EnumSet}$ runs in expected $\widetilde{O}(\sqrt{n})$ time. It satisfies Definitions A1 and A2 assumming privately-puncturable PRFs (that satisfy Definitions E2, E1, E3).

3 Preliminary 2PIR+ Protocol

We first design a preliminary 2PIR+ protocol (Figure 1) that helps with the exposition of our final protocol. In this preliminary 2PIR+ protocol the client has linear local storage and the communication is amortized $\widetilde{O}(\sqrt{n})$. Later, we will convert this 2PIR+ scheme into a space and communication-efficient 2PIR+ protocol (by using our PRS primitive of Section 4) that will yield our final 1PIR scheme. Crucially, the analysis of the preliminary protocol is almost the same as that of our final PIR protocol in Section 5.

Overview of our preliminary protocol. Our preliminary protocol works as follows. During the *preprocessing* phase, the client constructs a collection T of $\ell = \sqrt{n}\log^3 n$ "primary" sets and a collection Z of an additional $\sqrt{n}$ "reserve" sets. All sets are sampled from a fixed distribution $\mathbb{D}_n$ over the domain $\{0, \ldots, n-1\}$. While we can use any distribution for our preliminary protocol, we use a specific one that will serve the use of PRSs in Section 5. Both T and Z are sent to $\mathbf{server}_1$ and client receives the hints back, as explained in the introduction. Client stores locally the collections T and Z along with the hints. This is the main difference with our final protocol, where we will be storing keys instead of the sets themselves. To query an index x during the *query* phase, the

client finds some $T_j = (S_j, p_j)$ in T such that that S_j contains x, "removes" x and sends the new set to **server**$_2$. Then **server**$_2$ computes the parity of the new set and sends the parity back, at which point the client can compute $\mathsf{DB}[x]$, by xoring **server**$_2$'s response with p_j. As we will see, element removal in this context means resampling the membership of x via a `Resample` algorithm introduced below. To ensure the set distribution of T does not change across queries, our protocol has a *refresh* phase, where element x is "added", to the next available reserve set, via an `Add` algorithm introduced below. The protocol allows for $\sqrt{n}$ queries and achieves amortized sublinear server time over these $\sqrt{n}$ queries. After $\sqrt{n}$ queries, we re-run the offline phase.

The above protocol can fail with constant probability, as we will analyze in Lemma 31 below. To avoid this, as we indicate at the top of Figure 1, we run $\log n \log\log n$ parallel instances of the protocol and take the majority bit as the output answer. We now continue with the detailed description of the building blocks (such as algorithms `Resample` and `Add`) that our protocol uses.

Sampling distribution $\mathbb{D}_n$. For our preliminary protocol we are using the same distribution as the one induced by the PRS construction by Shi et al. [36] described in Section 2. This will help us seamlessly transition to our space-efficient protocol in Section 5. To sample a set S with elements from the domain $\{0,\ldots,n-1\}$ we define, for all $x \in \{0,\ldots,n-1\}$,

$$x \in S \Leftrightarrow \texttt{RO}(z[i:]) = 1 \text{ for all } i \in [0,m]\,,$$

where we recall that $m = \log n/2 + B, B = 2\log\log n$ and $z = 0^B||x$. Also, $\texttt{RO} : \{0,1\}^* \to \{0,1\}$ denotes a random oracle. We use the random oracle for exposition only—our final construction does not need one. Note for our preliminary protocol, the adversary cannot call the `RO` function or otherwise all the sets would revealed. We also define $\mathbb{D}_n^x$ to be a distribution where a set S is sampled from $\mathbb{D}_n$ *until* $x \in S$.

Functions with respect to $\mathbb{D}_n$. We define two functions with respect to the distribution $\mathbb{D}_n$ —these functions will be needed to describe our preliminary scheme. To define these functions, we first introduce what it means for two elements to be *related*.

Definition 31. *Function* $\texttt{Related}(x,y)$*, where* $x,y \in \{0,\ldots,n-1\}$*, returns a bit* $b \in \{0,1\}$ *where* $b = 1$ *(in which case we say that* x *is* related *to* y*) iff* x *and* y *share a suffix of length* $> \log n/2$ *in their binary representation.*

For example $\texttt{Related}(1000001, 1100001) = 1$ and $\texttt{Related}(1000001, 1101111) = 0$. Equipped with this, we define our two functions.

- $\texttt{Resample}(S,x) \to S'$: Given $x \in S$ as input, define $z = 0^B||x$. We sample a uniform bit for each suffix of z, $z[i:]$, for $i \in [0,m]$. For each $y \in S$ such that $\texttt{Related}(x,y)$ (including x), we check if any suffix of y was mapped to 0, and if so, remove it from S and return this new set.
- $\texttt{Add}(S,x) \to S'$: This function essentially "reprograms" the random oracle such that $\texttt{RO}(z[i:]) = 1$ for all $i \in [0,m]$, where $z = 0^B||x$. This may also affect membership of other elements $y \in \{0,\ldots,n-1\}$ that are not in S, but related to x with some probability. For us it will suffice that for most of executions, $\texttt{Add}(S,x) = S \cup \{x\}$. We bound the probability of this formally in Appendix B.

• Run $\omega(\log \lambda)$ instances of the protocol below.
• Output the majority bit maj in Step 4 of Query.
• Use maj as $\mathsf{DB}[x]$ in Step 2 of Refresh.

Offline phase: Preprocessing

1. **client** samples $\ell + \sqrt{n}$ sets from $\mathbb{D}_n$, $S_1, \ldots, S_{\ell+\sqrt{n}}$, where $\ell = \sqrt{n}\log^3 n$.
2. **client** sends sets $S_1, \ldots, S_{\ell+\sqrt{n}}$ to **server**$_1$ and **server**$_1$ returns a set of bits $p_1, \ldots, p_{\ell+\sqrt{n}}$, where
$$p_i = \oplus_{j \in S_i} \mathsf{DB}[j]\,.$$
3. **client** stores pairs of sets/hints
$$\mathsf{T} = \{T_j = (S_j, p_j)\}\,, \mathsf{Z} = \{Z_k = (S_k, p_k)\}\,,$$
where $j \in [\ell]$ and $k \in [\ell+1, \ell+\sqrt{n}]$.

Online phase: Query (input is index $x \in \{0, \ldots, n-1\}$)

1. **client** finds the first $T_j = (S_j, p_j)$ in T such that $x \in S_j$. If such T_j is not found, set $j = |\mathsf{T}| + 1$ and $T_j = (S_j, p_j)$ where $S_j \sim \mathbb{D}_n$ and p_j is uniform bit.
2. **client** sends $S' = \texttt{Resample}(S_j, x)$ to **server**$_2$.
3. **server**$_2$ returns $r = \bigoplus_{k \in S'} \mathsf{DB}[k]$.
4. **client** computes $\mathsf{DB}[x] = r \oplus p_j$.

Online phase: Refresh (executed when $j \leq |\mathsf{T}|$)

1. Let $Z_0 = (S_0, p_0)$ be the first item from Z.
2. Let $S_0^* = \texttt{Add}(S_0, x)$, and
$$p_0^* = p_0 \oplus (\mathsf{DB}[x] \wedge (x \notin S_0))\,.$$
3. **client** sets $T_j = (S_0^*, p_0^*)$, where T_j was consumed earlier, and removes Z_0 from Z.

Fig. 1. Our preliminary 2PIR+ protocol. With n we denote the size of DB and $[\ell] = [1, \ell]$.

Efficiency analysis. Our preliminary protocol in Figure 1 is inefficient: The online server time is $\widetilde{O}(\sqrt{n})$, client storage and computation is $\widetilde{O}(n)$ and bandwidth is $\widetilde{O}(\sqrt{n})$. It supports $\sqrt{n}$ queries, after which we need to re-run the offline phase.

Correctness proof. As we mentioned before, our basic protocol without parallel instances, has constant failure probability, less than $1/2$. We prove this through Lemma 31.

Lemma 31 (Correctness of protocol with no repetitions)**.** *Consider the protocol of Figure 1 with no repetitions and fix a query x_i. The probability that the returned bit $\mathsf{DB}[x_i]$ in Step 4 of* Query *is incorrect, assuming $\mathsf{DB}[x_{i-1}]$ used in Step 2 of* Refresh *is correct with overwhelming probability, is less than $1/2$.*

We give an overview of the intuition of the proof here and defer the full proof of Lemma 31 to Appendix B. We distinguish two cases. For the first query x_1, there are three cases where our protocol can fail. The first failure occurs if we cannot find an index j in T such that $x \in S_j$ for $T_j = (S_j, p_j)$ (Step 1 of Query). We can bound this failure by $1/n$. The second failure occurs when our `Resample` function does not remove x. This happens with probability $1/\sqrt{n}\log^2 n$. The third failure case occurs when we remove x, but also remove an element other than x within `Resample`. This can bounded by $1/2\log n$.

For every other query x_i, i greater than 1, we must consider an additional failure case which occurs when, in the Refresh phase, we add an element other than x within `Add`—which we can also bound by $1/2\log n$. Computing the final bound requires more work. It requires showing that Refresh only incurs a very small additional error probability to subsequent queries, which can also be bounded at the query step. We argue this formally in our proof of Theorem 31.

Amplifying correctness via repetition. To increase correctness of our scheme, we run k parallel instances of our protocol and set the output bit in Step 3 of Query to equal the majority of $\mathsf{DB}[x]$ over these k instance. We run Refresh with the correct $\mathsf{DB}[x]$ computed in Query so that we can apply Lemma 31. Let C be the event, where, over k instances of our preliminary PIR scheme, more than $\frac{k}{2}$ instances output the correct $\mathsf{DB}[x]$. Using a standard lower-tail Chernoff bound, we have that, if $p > 1/2$ is the probability $\mathsf{DB}[x]$ is correct, C's probability $> 1 - \exp(-\frac{1}{2p}k(p-\frac{1}{2})^2)$ which is overwhelming for $k = \omega(\log n)$, satisfying Definition 22. The same technique is used in our final PIR scheme.

Privacy proof. We now show that our preliminary PIR protocol satisfies privacy, per Definition 23. Proving privacy relies on two properties we define below. Both proofs are similar, so we provide only the proof of the less intuitive *Property 2*.

Property 1: Let $S \sim \mathbb{D}_n^x$ and $S' \sim \mathbb{D}_n$. Then $\texttt{Resample}(S, x)$ and S' are statistically indistinguishable.

Property 2: Let $S \sim \mathbb{D}_n$ and $S' \sim \mathbb{D}_n^x$. Then $\texttt{Add}(S, x)$ and S' are statistically indistinguishable.

Lemma 32. *Property 2 holds.*

Proof. Consider the set $S' \sim \mathbb{D}_n^x$ and the set S'' output as (i) $S \sim \mathbb{D}_n$; (ii) $S'' \leftarrow \texttt{Add}(S, x)$. For an arbitrary y in the domain we show that $\Pr[y \in S'] = \Pr[y \in S'']$. Recall $m = 1/2\log n + B$. We distinguish two cases.

1. y is not related to x.
   - **Computing** $\Pr[y \in S']$**.** Let F_i be the event that set S' is output in the i-th try, where $i = 1, 2, \ldots, \infty$. It is

$$\Pr[y \in S'] = \sum_{i=1}^{\infty} \Pr[y \in S' | F_i] \Pr[F_i] = \frac{1}{2^m} \sum_{i=1}^{\infty} \Pr[F_i] = \frac{1}{2^m}\,.$$

     In the above, $\Pr[y \in S' | F_i] = 1/2^m$ since x being in S' **does not affect** y's membership. Therefore for y to be a member, all m membership-test suffixes of y must evaluate to 1 during the i-th try, hence the derived probability.

- **Computing** $\Pr[y \in S'']$**.** Since adding x to S after S is sampled from $\mathbb{D}_n$ **does not affect** y's membership, it is $\Pr[y \in S''] = 1/2^m$.

2. y is related to x. Assume there are k (out of m) shared membership-test suffixes of x and y.
   - **Computing** $\Pr[y \in S']$**.** Again, let F_i be the event that set S' is output in the i-th try, where $i = 1, 2, \ldots, \infty$. It is

$$\Pr[y \in S'] = \sum_{i=1}^{\infty} \Pr[y \in S' | F_i] \Pr[F_i] = \frac{1}{2^{m-k}} \sum_{i=1}^{\infty} \Pr[F_i] = \frac{1}{2^{m-k}} .$$

     In the above, $\Pr[y \in S' | F_i] = 1/2^{m-k}$. This is because x being in S' **does affect** y's membership. Therefore for y to be a member, all remaining $m - k$ membership-test suffixes of y must evaluate to 1 during the i-th try, hence the derived probability.
   - **Computing** $\Pr[y \in S'']$**.** Adding x to S after S is sampled from $\mathbb{D}_n$ sets k membership-test suffixes of y to 1. Therefore for y to be a member of S'', the remaining membeship-test suffixes have to be set to 1 before x is added, meaning $\Pr[y \in S''] = 1/2^{m-k}$.

Therefore the distributions are identical. □

Given these two properties, our proof sketch goes as follows. For the first query, we pick an entry $T_j = (S_j, p_j)$ from T whose S_j contains the index x we want to query. Since S_j is the first set in T to contain x, $S_j \sim \mathbb{D}_n^x$. By Property 1, since what **server**$_2$ sees is $S' = \texttt{Resample}(S_j, x)$, S' is indistinguishable from a random set drawn from $\mathbb{D}_n$, and therefore, the query reveals nothing about the query index x to **server**$_2$.

For every other query, we argue that the Refresh step maintains the distribution of T. Note that after a given set S_j is used, re-using it for the same query *or* a different query could create privacy problems. That is why after each query, we must replace S_j with an identically distributed set. By Property 2, S_j and $\texttt{Add}(S_0, x)$ are identically distributed. Then, the swap maintains the distribution of sets in T and therefore the view of **server**$_2$ is also simulatable without x. These arguments form the crux of the proof of Theorem 31; we provide the full proof in Appendix B.

Theorem 31 (Preliminary 2PIR+ protocol). *The 2PIR+ scheme in Figure 1 is correct (per Definition 22) and private (per Definition 23) and has: (i) $\widetilde{O}(n)$ client storage $\widetilde{O}(n)$ client time; (ii) $\widetilde{O}(\sqrt{n})$ amortized server time and no additional server storage; (iii) $\widetilde{O}(\sqrt{n})$ amortized bandwidth.*

4 Adaptable Pseudorandom Sets

In this section, we introduce the main primitive required for achieving our result, an *adaptable pseudorandom set*. The main difference from a privately-puncturable PRS introduced in Section 2 is the support for the "add" procedure, as well as any logarithmic (in the set size) number of additions or removals, as opposed to a single removal. This will eventually allow us to port the protocol from Section 3 into a 1PIR protocol that

has much improved complexities, such as sublinear client storage and polylogarithmic communication. We now give the formal definition and then we present a construction that satisfies our definition.

Definition 41 (Adaptable PRS)**.** *An adaptable PRS has five algorithms: (i)* $\mathsf{Gen}(1^\lambda, n) \to (msk, sk)$*: Outputs our set's key* sk *and master key* msk*, given security parameter* λ *and set domain* $\{0, \ldots, n-1\}$*; (ii)* $\mathsf{EnumSet}(sk) \to S$*: Outputs set* S *given* sk*; (iii)* $\mathsf{InSet}(sk, x) \to b$*: Outputs bit* 1 *iff* $x \in \mathsf{EnumSet}(sk)$*; (iv)* $\mathsf{Resample}(msk, sk, x) \to sk_x$*: Outputs secret key* sk_x *that corresponds to an updated version of the set (initially generated by* sk*) after element* x *is resampled; (v)* $\mathsf{Add}(msk, sk, x) \to sk^x$*: Outputs secret key* sk^x *that corresponds to an updated version of the set (initially generated by* sk*) after element* x *is added.*

Note that our interface differs from privately-puncturable PRSs introduced in Section 2 in that our resample and add operations are dependent on both msk and sk; we will see why below.

Security definitions for adaptable PRSs. Our adaptable PRS must satisfy five definitions. Three of them, *functionality preservation with respect to resampling*, *pseudorandomness with respect to a distribution* $\mathbb{D}_n$ and *security in resampling* are identical to the equivalent definitions from privately-puncturable PRSs, namely Definitions A3, A1, A2 in Appendix A. We give two additional definitions in Appendix A (definitions A5 and A4) that relate to addition. First, *functionality preservation with respect to addition*, meaning that adding always yields a superset of the original set and can only cause elements related to x (which are few) to be added to the set. Second, *security in addition*, meaning that generating fresh keys until we find one where x belongs to the set is equivalent to generating one fresh key and then adding x into it.

Intuition of our construction: Introduce an additional key. Our core idea is to use two keys $sk[0]$ and $sk[1]$ and define the evaluation on the suffixes that determines membership as the XOR of $F.\mathsf{Eval}(sk[0], \cdot)$ and $F.\mathsf{Eval}(sk[1], \cdot)$. In this way, we can add to one key, and resample the other, independently. Note that this idea can support any fixed number of additions or resamplings (removals), by adding extra PRF keys. This simple construction circumvents many problems related with trying to perform multiple operations on the same key. Each key has one well defined operation. This also makes showing security and privacy straight-forward to argue.

We present a summary of our construction below. The detailed implementation is in Figure 3 in Appendix C.

Key generation. Let F be a privately-puncturable PRF. For key generation, we run $F.\mathsf{Gen}$ twice, outputting $msk[0]$ and $msk[1]$. After puncturing on m "useless" points (for reasons we explained in Section 2), we output $sk[0]$ and $sk[1]$. And finally we output $sk = (sk[0], sk[1])$ and $msk = (msk[0], msk[1])$.

Set membership and enumeration. For each $x \in \{0, \ldots, n-1\}$ we define

$$x \in S \Leftrightarrow F.\mathsf{Eval}(sk[0], z[i:]) \oplus F.\mathsf{Eval}(sk[1], z[i:]) = 1 \text{ for all } i \in [0, m]\,,$$

where we recall $m = \log n/2 + B, B = 2\log\log n$ and $z = 0^B||x$. For enumeration, we use the same algorithm as Shi et al. [36], with the difference that evaluation is done as the XOR of two evaluations, as above.

Resampling. Resampling works exactly as resampling in privately-puncturable PRSs (by calling $F.\mathsf{Puncture}$) and uses, without loss of generality, $msk[1]$ as input. The output replaces only the second part of sk—thus we require sk as input so that we can output the first part intact.

Addition. To add an element x, we call $F.\mathsf{Puncture}$ on input $msk[0]$, and then check x's membership on the punctured key. If x was added, we output the punctured key, else, we try puncturing from the master key again, until x is resampled *into* the set. This is the reason why it is necessary to have a rerandomizable puncture operation. Naively, this algorithm takes $\widetilde{O}(\sqrt{n})$ time, but we show in the Appendix how to reduce this to $\widetilde{O}(1)$ by leveraging the puncturable PRF used. Our final theorem is Theorem 41, and the construction and proof can be found in Appendix C.

Theorem 41 (Adaptable PRS construction). *Assuming* LWE*, the scheme in Figure 3 satisfies correctness, pseudorandomness with respect to $\mathbb{D}_n$ (Definition A1), functionality preservation in resampling and addition (Definitions A3 and A5), security in resampling and addition (Definitions A2 and A4), and has the following complexities: (i) keys sk and msk have $\widetilde{O}(1)$ size; (ii) membership testing, resampling and addition take $\widetilde{O}(1)$ time; (iii) enumeration takes $\widetilde{O}(\sqrt{n})$ time.*

5 More Efficient 2PIR+ and Near-Optimal 1PIR

We now use adaptable PRSs introduced in the previous section to build a more efficient 2PIR+ scheme (one with $\widetilde{O}(\sqrt{n})$ client storage and $\widetilde{O}(1)$ communication complexity) which can be compiled, using FHE, into a 1PIR scheme with the same complexities, as we explained in the introduction. The main idea is to replace the actual sets, stored by the client in their entirety in our preliminary protocol, with PRS keys that support succinct representation, addition and removal. In particular, our proposed protocol in Figure 2 is identical to our preliminary protocol in Figure 1 except for the following main points: (i) In the offline phase, instead of sampling sets from $\mathbb{D}_n$, we generate keys (msk, sk) for adaptable PRSs that correspond to sets of the same distribution $\mathbb{D}_n$. (ii) In the online phase, we run `Resample` and `Add` defined in the adaptable PRS. These have exactly the same effect in the output set, except the operations are done on the set key not the set. (iii) We can check membership efficiently using InSet. We now introduce Theorem 51.

Theorem 51 (Efficient 2PIR+ protocol). *Assuming* LWE*, the 2PIR+ scheme in Figure 2 is correct (per Definition 22) and private (per Definition 23) and has: (i) $\widetilde{O}(\sqrt{n})$ client storage and $\widetilde{O}(\sqrt{n})$ client time; (ii) $\widetilde{O}(\sqrt{n})$ amortized server time and no additional server storage; (iii) $\widetilde{O}(1)$ amortized bandwidth.*

Unlimited queries. Our scheme can handle $\sqrt{n}$ queries but can be extended to unlimited queries: We just rerun the offline phase after all secondary sets are used. This maintains the complexities from Theorem 51.

• Run $\omega(\log \lambda)$ instances of the protocol below.
• Output the majority bit maj in Step 4 of Query.
• Use maj as $\mathsf{DB}[x]$ in Step 2 of Refresh.

Offline phase: Preprocessing

1. **client** generates $\ell + \sqrt{n}$ PRSet keys

$$(msk_1, sk_1), \ldots, (msk_{\ell+\sqrt{n}}, sk_{\ell+\sqrt{n}})$$

with $\mathsf{Gen}(1^\lambda, n)$, $\ell = \sqrt{n}\log^3 n$.
2. **client** sends keys $sk_1, \ldots, sk_{\ell+\sqrt{n}}$ to **server**$_1$ and **server**$_1$ returns a set of bits $p_1, \ldots, p_{\ell+\sqrt{n}}$, where

$$p_i = \oplus_{j \in \mathsf{EnumSet}(sk_i)} \mathsf{DB}[j]\,.$$

3. **client** stores pairs of keys/hints

$$\mathsf{T} = \{T_j = (msk_j, sk_j, p_j)\}\,, \mathsf{Z} = \{Z_k = (msk_k, sk_k, p_k)\}\,,$$

where $j \in [\ell]$ and $k \in [\ell+1, \ell+\sqrt{n}]$.

Online phase: Query (input is index $x \in \{0, n-1\}$)

1. **client** finds the first $T_j = (msk_j, sk_j, p_j)$ in T such that $\mathsf{InSet}(sk_j, x) = 1$. If such T_j is not found, set $j = |\mathsf{T}| + 1$ and $T_j = (msk_j, sk_j, p_j)$ where $\mathsf{Gen}(1^\lambda, n) \to (msk_j, sk_j)$ and p_j is uniform bit.
2. **client** sends $sk' \leftarrow \mathsf{Resample}(msk_j, sk_j, x)$ to **server**$_2$.
3. **server**$_2$ returns $r = \bigoplus_{k \in \mathsf{EnumSet}(sk')} \mathsf{DB}[k]$.
4. **client** computes $\mathsf{DB}[x] = r \oplus p_j$.

Online phase: Refresh (executed when $j \le |\mathsf{T}|$)

1. Let $Z_0 = (msk_0, sk_0, p_0)$ be the first item from Z.
2. Let $(msk_0^x, sk_0^*) \leftarrow \mathsf{Add}(msk_0, sk_0, x)$ and

$$p_0^* = p_0 \oplus (\mathsf{DB}[x] \wedge (\neg \mathsf{InSet}(x, sk_0)))\,.$$

3. **client** sets $T_j = (msk_0^x, sk_0^*, p_0^*)$, where T_j was consumed earlier, and removes Z_0 from Z.

Fig. 2. Our 2PIR+ for n-bit DB using adaptable PRS (Gen, $\mathsf{EnumSet}$, InSet, $\mathsf{Resample}$, Add).

Trade-offs in client space and server time. Our scheme enjoys a trade-off between client space and server time. One can increase the number of elements of each PRSet to n/Q. This would change the number of sets required for our scheme to Q, and consequently our scheme would enjoy Q client space, at the expense of requiring n/Q online server time. This tradeoff holds in the other direction as well (increasing client

space reduces online server time). In any case, the product of client space and online server time must equal n, as shown by Corrigan-Gibbs et al. in [16].

From 2PIR+ to 1 PIR with same complexities. As detailed in [15], we can port our 2PIR+ to 1PIR by merging $\mathbf{server}_1$ and $\mathbf{server}_2$ and executing the work of $\mathbf{server}_1$ using FHE. We require a symmetric key FHE scheme that is *gate-by-gate* [15], where *gate-by-gate* means that encrypted evaluation runs in time $\widetilde{O}(|C|)$ for a circuit of size $|C|$. As noted in [15], this is a property of standard FHE based on LWE [9, 26]. With this, we can use a batch parity Boolean circuit C that, given a database of size n and l lists of size m, C computes the parity of the lists in $\widetilde{O}(l \cdot m + n)$ time [15]. The last consideration is how to perform the set evaluation under FHE. This can be done using slight modifications to our evaluation algorithm and using oblivious sorting. We discuss this further in Appendix D. Our main result, Theorem 52, is as follows.

Theorem 52 (Near-Optimal 1PIR protocol). *Assuming* LWE*, there exists an 1PIR scheme that is correct (per Definition 22) and private (per Definition 23) and has: (i)* $\widetilde{O}(\sqrt{n})$ *client storage and* $\widetilde{O}(\sqrt{n})$ *client time; (ii)* $\widetilde{O}(\sqrt{n})$ *amortized server time and no additional server storage; (iii)* $\widetilde{O}(1)$ *amortized bandwidth.*

Our proof for both theorems introduced are located in Appendix D, but follow closely from our adaptable pseudorandom set properties and the proof from our preliminary protocol, along with the tools introduced above.

Acknowledgements

This work was supported by the NSF, VMware and Protocol Labs.

References

1. Angel, S., Chen, H., Laine, K., Setty, S.: PIR with Compressed Queries and Amortized Query Processing. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 962–979 (May 2018). https://doi.org/10.1109/SP.2018.00062, iSSN: 2375-1207
2. Angel, S., Setty, S.: Unobservable communication over fully untrusted infrastructure. In: Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation. pp. 551–569. OSDI'16, USENIX Association, USA (Nov 2016)
3. Backes, M., Kate, A., Maffei, M., Pecina, K.: ObliviAd: Provably Secure and Practical Online Behavioral Advertising. In: 2012 IEEE Symposium on Security and Privacy. pp. 257–271 (May 2012). https://doi.org/10.1109/SP.2012.25, iSSN: 2375-1207
4. Beimel, A., Ishai, Y.: Information-Theoretic Private Information Retrieval: A Unified Construction. In: Goos, G., Hartmanis, J., van Leeuwen, J., Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) Automata, Languages and Programming, vol. 2076, pp. 912–926. Springer Berlin Heidelberg, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-48224-5_74, http://link.springer.com/10.1007/3-540-48224-5_74, series Title: Lecture Notes in Computer Science
5. Beimel, A., Ishai, Y., Malkin, T.: Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing. In: Bellare, M. (ed.) Advances in Cryptology — CRYPTO 2000. pp. 55–73. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_4

6. Bell, J.H., Bonawitz, K.A., Gascón, A., Lepoint, T., Raykova, M.: Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1253–1269. CCS '20, Association for Computing Machinery, New York, NY, USA (Oct 2020). https://doi.org/10.1145/3372297.3417885, https://doi.org/10.1145/3372297.3417885
7. Boneh, D., Kim, S., Montgomery, H.: Private Puncturable PRFs from Standard Lattice Assumptions. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 415–445. Lecture Notes in Computer Science, Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_15
8. Brakerski, Z., Tsabary, R., Vaikuntanathan, V., Wee, H.: Private Constrained PRFs (and More) from LWE. In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography. pp. 264–302. Lecture Notes in Computer Science, Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_10
9. Brakerski, Z., Vaikuntanathan, V.: Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Rogaway, P. (eds.) Advances in Cryptology – CRYPTO 2011, vol. 6841, pp. 505–524. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_29, http://link.springer.com/10.1007/978-3-642-22792-9_29, series Title: Lecture Notes in Computer Science
10. Canetti, R., Chen, Y.: Constraint-Hiding Constrained PRFs for NC^{1}from LWE. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 446–476. Lecture Notes in Computer Science, Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_16
11. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. pp. 41–41. IEEE Computer Society (Oct 1995). https://doi.org/10.1109/SFCS.1995.492461, https://www.computer.org/csdl/proceedings-article/focs/1995/71830041/12OmNzYNNfi, iSSN: 0272-5428
12. Chor, B., Gilboa, N.: Computationally private information retrieval (extended abstract). In: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. pp. 304–313. STOC '97, Association for Computing Machinery, New York, NY, USA (May 1997). https://doi.org/10.1145/258533.258609, https://dl.acm.org/doi/10.1145/258533.258609
13. Chor, B., Gilboa, N., Naor, M.: Private Information Retrieval by Keywords (1998), https://eprint.iacr.org/1998/003, report Number: 003
14. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. Journal of the ACM **45**(6), 965–981 (Nov 1998). https://doi.org/10.1145/293347.293350, https://doi.org/10.1145/293347.293350
15. Corrigan-Gibbs, H., Henzinger, A., Kogan, D.: Single-Server Private Information Retrieval with Sublinear Amortized Time. In: Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part II. pp. 3–33. Springer-Verlag, Berlin, Heidelberg (May 2022). https://doi.org/10.1007/978-3-031-07085-3_1, https://doi.org/10.1007/978-3-031-07085-3_1
16. Corrigan-Gibbs, H., Kogan, D.: Private Information Retrieval with Sublinear Online Time. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 44–

75. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_3
17. Devadas, S., van Dijk, M., Fletcher, C.W., Ren, L., Shi, E., Wichs, D.: Onion ORAM: 13th International Conference on Theory of Cryptography, TCC 2016. Theory of Cryptography - 3th International Conference, TCC 2016-A, Proceedings pp. 145–174 (2016). https://doi.org/10.1007/978-3-662-49099-0_6, http://www.scopus.com/inward/record.url?scp=84954156296&partnerID=8YFLogxK, publisher: Springer
18. Di Crescenzo, G., Ishai, Y., Ostrovsky, R.: Universal Service-Providers for Private Information Retrieval. Journal of Cryptology **14**(1), 37–74 (Jan 2001). https://doi.org/10.1007/s001450010008, http://link.springer.com/10.1007/s001450010008
19. Di Crescenzo, G., Malkin, T., Ostrovsky, R.: Single Database Private Information Retrieval Implies Oblivious Transfer. In: Preneel, B. (ed.) Advances in Cryptology — EUROCRYPT 2000. pp. 122–138. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2000). https://doi.org/10.1007/3-540-45539-6_10
20. Dong, C., Chen, L.: A Fast Single Server Private Information Retrieval Protocol with Low Communication Cost. In: Kutyłowski, M., Vaidya, J. (eds.) Computer Security - ESORICS 2014, vol. 8712, pp. 380–399. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-11203-9_22, http://link.springer.com/10.1007/978-3-319-11203-9_22, series Title: Lecture Notes in Computer Science
21. Dvir, Z., Gopi, S.: 2-Server PIR with Subpolynomial Communication. Journal of the ACM **63**(4), 1–15 (Nov 2016). https://doi.org/10.1145/2968443, https://dl.acm.org/doi/10.1145/2968443
22. Efremenko, K.: 3-Query Locally Decodable Codes of Subexponential Length. SIAM Journal on Computing **41**(6), 1694–1703 (Jan 2012). https://doi.org/10.1137/090772721, http://epubs.siam.org/doi/10.1137/090772721
23. Garg, S., Mohassel, P., Papamanthou, C.: TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016. pp. 563–592. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_20
24. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09. p. 169. ACM Press, Bethesda, MD, USA (2009). https://doi.org/10.1145/1536414.1536440, http://portal.acm.org/citation.cfm?doid=1536414.1536440
25. Gentry, C., Ramzan, Z.: Single-Database Private Information Retrieval with Constant Communication Rate. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Automata, Languages and Programming, vol. 3580, pp. 803–815. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11523468_65, http://link.springer.com/10.1007/11523468_65, series Title: Lecture Notes in Computer Science
26. Gentry, C., Sahai, A., Waters, B.: Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology – CRYPTO 2013. pp. 75–92. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_5

27. Goldreich, O., Goldwasser, S., Micali, S.: How to Construct Random Functions (Extended Abstract). In: FOCS (1984). https://doi.org/10.1109/SFCS.1984.715949
28. Gupta, T., Crooks, N., Mulhern, W., Setty, S., Alvisi, L., Walfish, M.: Scalable and private media consumption with Popcorn. In: Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation. pp. 91–107. NSDI'16, USENIX Association, USA (Mar 2016)
29. Kazama, K., Kamatsuka, A., Yoshida, T., Matsushima, T.: A Note on a Relationship between Smooth Locally Decodable Codes and Private Information Retrieval. In: 2020 International Symposium on Information Theory and Its Applications (ISITA). pp. 259–263 (Oct 2020), iSSN: 2689-5854
30. Kiayias, A., Leonardos, N., Lipmaa, H., Pavlyk, K., Tang, Q.: Optimal Rate Private Information Retrieval from Homomorphic Encryption. Proceedings on Privacy Enhancing Technologies **2015**(2), 222–243 (Jun 2015). https://doi.org/10.1515/popets-2015-0016, https://www.sciendo.com/article/10.1515/popets-2015-0016
31. Kogan, D., Corrigan-Gibbs, H.: Private Blocklist Lookups with Checklist. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 875–892. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/kogan
32. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: single database, computationally-private information retrieval. In: Proceedings 38th Annual Symposium on Foundations of Computer Science. pp. 364–373. IEEE Comput. Soc, Miami Beach, FL, USA (1997). https://doi.org/10.1109/SFCS.1997.646125, http://ieeexplore.ieee.org/document/646125/
33. Lipmaa, H.: An oblivious transfer protocol with log-squared communication. In: Proceedings of the 8th international conference on Information Security. pp. 314–328. ISC'05, Springer-Verlag, Berlin, Heidelberg (Sep 2005). https://doi.org/10.1007/11556992_23, https://doi.org/10.1007/11556992_23
34. Lipmaa, H., Pavlyk, K.: A Simpler Rate-Optimal CPIR Protocol. In: Financial Cryptography and Data Security, 2017 (2017), http://eprint.iacr.org/2017/722
35. Mughees, M.H., Chen, H., Ren, L.: OnionPIR: Response Efficient Single-Server PIR. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2292–2306. CCS '21, Association for Computing Machinery, New York, NY, USA (Nov 2021). https://doi.org/10.1145/3460120.3485381, https://doi.org/10.1145/3460120.3485381
36. Shi, E., Aqeel, W., Chandrasekaran, B., Maggs, B.: Puncturable Pseudorandom Sets and Private Information Retrieval with Near-Optimal Online Bandwidth and Time. In: Advances in Cryptology - CRYPTO (2021), http://eprint.iacr.org/2020/1592
37. Singanamalla, S., Chunhapanya, S., Hoyland, J., Vavruša, M., Verma, T., Wu, P., Fayed, M., Heimerl, K., Sullivan, N., Wood, C.: Oblivious DNS over HTTPS (ODoH): A Practical Privacy Enhancement to DNS. Proceedings on Privacy Enhancing Technologies **2021**(4), 575–592 (Oct 2021). https://doi.org/10.2478/popets-2021-0085, https://www.sciendo.com/article/10.2478/popets-2021-0085
38. Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. Journal of the ACM **55**(1), 1–16 (Feb 2008). https://doi.org/10.1145/1326554.1326555, https://dl.acm.org/doi/10.1145/1326554.1326555
39. Yekhanin, S.: Locally Decodable Codes and Private Information Retrieval Schemes. Information Security and Cryptography, Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14358-8, http://link.springer.com/10.1007/978-3-642-14358-8

40. Zhou, M., Lin, W.K., Tselekounis, Y., Shi, E.: Optimal Single-Server Private Information Retrieval. ePrint IACR (2022)

A Definitions

A.1 Additional Definitions for Adaptable PRSs

Our adaptable PRS primitive will satisfy the following definitions.

Definition A1 (Pseudorandomness with respect to some distribution $\mathbb{D}_n$ for privately-puncturable PRSs [36])**.** *A privately-puncturable PRS scheme (*Gen*,* $\mathsf{EnumSet}$*,* InSet*,* $\mathsf{Resample}$*) satisfies* pseudorandomness with respect to some distribution $\mathbb{D}_n$ *if the distribution of* $\mathsf{EnumSet}(sk)$*, where* sk *is output by* $\mathsf{Gen}(\lambda, n)$*, is indistinguishable from a set sampled from* $\mathbb{D}_n$*.*

Definition A2 (Security in resampling for privately-puncturable PRSs [36])**.** *A privately-puncturable PRS scheme (*Gen*,* $\mathsf{EnumSet}$*,* InSet*,* $\mathsf{Resample}$*) satisfies* security in resampling *if, for any* $x \in \{1, \ldots, n-1\}$*, the following two distributions are computationally indistinguishable.*

- *Run* $\mathsf{Gen}(\lambda, n) \to (sk, msk)$*, output* sk*.*
- *Run* $\mathsf{Gen}(\lambda, n) \to (sk, msk)$ *until* $\mathsf{InSet}(sk, x) \to 1$*, output* $sk_x = \mathsf{Resample}(msk, x)$*.*

Definition A3 (Functionality preservation in resampling for privately-puncturable PRSs [36])**.** *We say that a privately-puncturable PRS scheme (*Gen*,* $\mathsf{EnumSet}$*,* InSet*,* $\mathsf{Resample}$*)* satisfies functionality preservation in resampling *with respect to a predicate* $\mathtt{Related}$ *if, with probability* $1 - \mathtt{negl}(\lambda)$ *for some negligible function* $\mathtt{negl}(.)$*, the following holds. If* $\mathsf{Gen}(1^\lambda, n) \to (sk, msk)$ *and* $\mathsf{Resample}(msk, x) \to sk_x$ *where* $x \in \mathsf{InSet}(sk)$ *then*

1. $\mathsf{EnumSet}(sk_x) \subseteq \mathsf{EnumSet}(sk)$*;*
2. $\mathsf{EnumSet}(sk_x)$ *runs in time no more than* $\mathsf{EnumSet}(sk)$*;*
3. *For any* $y \in \mathsf{EnumSet}(sk) \setminus \mathsf{EnumSet}(sk_x)$*, it must be that* $\mathtt{Related}(x, y) = 1$*.*

Definition A4 (Security in addition for adaptable PRSs)**.** *We say that an adaptable PRS scheme (*Gen*,* $\mathsf{EnumSet}$*,* InSet*,* $\mathsf{Resample}$*,* Add*) satisfies* security in addition *if, for any* $x \in \{0, \ldots, n-1\}$*, the following two distributions are computationally indistinguishable.*

- *Run* $\mathsf{Gen}(1^\lambda, n) \to (sk, msk)$ *until* $\mathsf{InSet}(sk, x) \to 1$*. Let* $msk[0] = null$ *and output* (msk, sk)*.*
- *Run* $\mathsf{Gen}(1^\lambda, n) \to (sk, msk)$*. Output* $(msk^x, sk^x) \leftarrow \mathsf{Add}(msk, sk, x)$*.*

Definition A5 (Functionality preservation in addition for adaptable PRS)**.** *We say that an adaptable PRS scheme* $(\mathsf{Gen}, \mathsf{EnumSet}, \mathsf{InSet}, \mathsf{Resample}, \mathsf{Add})$ *satisfies* functionality preservation in addition *with respect to a predicate* $\mathtt{Related}$ *if, with probability* $1 - \mathtt{negl}(\lambda)$ *for some negligible function* $\mathtt{negl}(.)$*, the following holds. If* $\mathsf{Gen}(1^\lambda, n) \to (sk, msk)$ *and* $\mathsf{Add}(msk, sk, x) \to sk^x$ *then*

- $\mathsf{EnumSet}(sk) \subseteq \mathsf{EnumSet}(sk^x)$*;*
- *For all* $y \in \mathsf{EnumSet}(sk^x) \setminus \mathsf{EnumSet}(sk)$ *it must be that* $\mathtt{Related}(x, y) = 1$*.*

B Correctness Lemmata

See below the proof of Lemma 31. We then use it to prove Theorem 31.

Proof. Recall that we fix $B = 2 \log \log n$. As alluded to in Section 3, we can split our failure probability in three cases:

- Case 1: x_i is not in any primary set that was preprocessed.
- Case 2: The resampling does not remove x_i.
- Case 3: Resampling removes *more* that just x_i from the set.

Case 1: We first note that, from our distribution $\mathbb{D}_n$, for any $x \in \{0, \ldots, n-1\}$, we have that, for $S \sim \mathbb{D}_n$,

$$\Pr[x \in S] = \left(\frac{1}{2}\right)^{\frac{1}{2}\log n + B} = \frac{1}{\sqrt{n}} \left(\frac{1}{2}\right)^{B} = \frac{1}{2^B \sqrt{n}} .$$

Then note that the expected size of S is the sum of the probability of each element being in the set, i.e.,

$$\mathbb{E}\left[|S|\right] = \mathbb{E}\left[\sum_{x=0}^{n-1} \frac{1}{2^B \sqrt{n}}\right] = \sum_{x=0}^{n-1} \mathbb{E}\left[\frac{1}{2^B \sqrt{n}}\right] = \frac{\sqrt{n}}{2^B} \leq \frac{\sqrt{n}}{(\log n)^2} .$$

We can conclude that the desired probability is

$$\Pr[x \notin \cup_{i \in [1,l]} S_i] = \left(1 - \frac{1}{\sqrt{n}(\log n)^2}\right)^{\sqrt{n}(\log n)^3} \leq \left(\frac{1}{e}\right)^{\log n} \leq \frac{1}{n} ,$$

where $\ell = \sqrt{n} \log^3 n$ and $S_1, \ldots, S_\ell \sim (\mathbb{D}_n)^\ell$.
Case 2: Assuming there is a set S such that $x_i \in S$, by construction of `Resample`, it is easy to see that the probability that x_i is not removed from S is equivalent to a Bernoulli variable that is 1 with probability $p = \frac{1}{\sqrt{n} \cdot 2^B}$, since we toss $1/2 \log n + B$ coins, and x is not removed only if all of these coins evaluate to 1. Therefore

$$\Pr[x_i \in \texttt{Resample}(S, x_i)] = \frac{1}{\sqrt{n} \cdot 2^B} \leq \frac{1}{\sqrt{n} \log^2 n} .$$

Case 3: Note that for any k less than $\log n$, there are exactly $2^{\log n - k} - 1$, or less than $2^{\log n - k}$ strings in $\{0,1\}^{\log n}$, that are different than x share a suffix of length $\geq k$ with x. Note that since x is in the set, for any k, the probability that a string y that has a common suffix of length exactly k with x is included in the set is the chance that its initial B bits *and* its remaining bits not shared with x evaluate to 1, namely, for any k less than $\log n$ and $y = \{0,1\}^{\log n - k} || x[\log n - k :]$ we have that:

$$\Pr[y \in S] = \frac{1}{2^B 2^{\log n - k}} .$$

Let N_k be the set of strings in the set that share a longest common suffix with x of length k. Then, since we know that there are at most $2^{\log n - k}$ such strings, we can say

that for any k, the expected size of N_k is

$$\mathbb{E}\left[|N_k|\right] \leq \mathbb{E}\left[\sum_{x=1}^{2^{\log n-k}} \frac{1}{2^B 2^{\log n-k}}\right] = \sum_{x=1}^{2^{\log n-k}} \mathbb{E}\left[\frac{1}{2^B 2^{\log n-k}}\right]$$
$$= 2^{\log n-k}\frac{1}{2^B 2^{\log n-k}} = \frac{1}{2^B}.$$

Then, for our construction, where we only check prefixes for k greater than $(1/2)\log n$, we can find that the sum of the expected size of N_k, for each such k is

$$\mathbb{E}\left[\sum_{k=\frac{1}{2}\log n+1}^{\log n-1} |N_k|\right] = \sum_{k=\frac{1}{2}\log n+1}^{\log n-1} \mathbb{E}\left[|N_k|\right] \leq \left(\frac{1}{2}\log n - 1\right)\frac{1}{2^B}$$
$$= \frac{\log n - 2}{2(\log n)^2} \leq \frac{1}{2\log n}.$$

Clearly, we can bound the probability of removing an element along with x_i by the probability that there exists a related element to x_i in the set, by previous discussion in Section 3. Then, given each bound above, assuming that the previous query was correct and that the refresh phase maintains the set distribution, we see that the probability that the returned bit $\mathsf{DB}[x_i]$ is incorrect for query step i is

$$\Pr[\mathsf{DB}[x_i] \text{ is incorrect}] \leq \frac{1}{n} + \frac{1}{\sqrt{n}\log^2 n} + \frac{1}{2\log n} \leq \frac{3}{2\log n} < \frac{1}{3},$$

for $n \geq 32$.

Now we introduce a new lemma that will help us prove Theorem 31. This lemma will bound the probability that `Add` does not work as expected. The intuition here is that, just like `Resample` can remove elements (already in the set) related to the resampled element, `Add` can add elements (not in the set) related to the added element. Below, we are bounding the number of elements *that are not* x and are expected to be added to the set when we add x. As we explained in Section 3, this is a "failure case", since it means that our set will not be what we expect.

Lemma B1 (Adding related elements)**.** *For $S \sim \mathbb{D}_n$, and any $x \in \{0,\ldots,n-1\}$, the related set $S_{almost,x}$ is defined as*

$$S_{almost,x} = \{y \mid y \in \texttt{Add}(S,x) \setminus (S \cup \{x\})\}.$$

Then the expected size of $S_{almost,x}$ is at most $\frac{1}{2\log n}$.

Proof. Note that for any k less than $\log n$, there are less than $2^{\log n-k}$ strings in $\{0,1\}^{\log n}$ that share a suffix of length greater than or equal to k with x that do not equal x. The probability that a string y that has a common suffix of exactly k with x is included in $S_{almost,x}$ is the chance that its initial B bits *and* its remaining bits not shared with x

evaluate to 1. Namely, let us say that

$$S_{almost,x} = \bigcup N_k \,,$$

for any $k \in \mathbb{N}$ that is less than $\log n$ and more than $(1/2)\log n$. We define each N_k as

$$N_k = \{y : y = \{0,1\}^{\log n - k} || x[\log n - k :]\} \,.$$

Since this is the same size as the N_k in Case 3 of Lemma 31, and we are iterating over the same k, the expected size of $S_{almost,x}$ is

$$\mathbb{E}\left[|S_{almost,x}|\right] \leq \frac{1}{2\log n} \,.$$

□

We are now equipped with all the tools we need to prove Theorem 31. We prove it below:

Proof. We first prove privacy of the scheme, then proceed to prove correctness. The asymptotics follow by construction and were argued in Section 3.
Privacy. Privacy for $\mathbf{server}_1$ is trivial. It only ever sees random sets generated completely independent of the queries and is not interacted with online. We present the privacy proof for $\mathbf{server}_2$ below.

Privacy with respect to $\mathbf{server}_2$, as per our definition, must be argued by showing there exists a stateful algorithm `Sim` that can run without knowledge of the query and be indistinguishable from an honest execution of the protocol, from the view of any PPT adversary $\mathcal{A}$ acting as $\mathbf{server}_2$ for any protocol $\mathbf{server}_1^*$ acting as $\mathbf{server}_1$. First, we note that the execution of the protocol between **client** and $\mathbf{server}_2$ is independent of **client**'s interaction with $\mathbf{server}_1$. **client** generates sets and queries $\mathbf{server}_1$ in the offline phase for their parity. Although this affects correctness of each query, it does not affect the message sent to $\mathbf{server}_2$ at each step of the online phase, since this is decided by the sets, generated by **client**. Then, we can rewrite our security definition, equivalently, disregarding **client**'s interactions with $\mathbf{server}_1$.

We want to show that for any query q_t for $t \in [1, Q]$, q_t leaks no information about the query index x_t to $\mathbf{server}_2$, or that interactions between **client** and $\mathbf{server}_2$ can be simulated with no knowledge of x_t. To do this, we show, equivalently, that the following two experiments are computationally indistinguishable.

- $\mathbf{Expt}_0$: Here, for each query index x_t that **client** receives, **client** interacts with $\mathbf{server}_2$ as in our PIR protocol.
- $\mathbf{Expt}_1$ In this experiment, for each query index x_t that **client** receives, **client** *ignores* x_t, samples a fresh $S \sim \mathbb{D}_n$ and sends S to $\mathbf{server}_2$.

First we define an intermediate experiment $\mathbf{Expt}_1^*$.

- $\mathbf{Expt}_1^*$: For each query index x_t that **client** receives, **client** samples $S \sim \mathbb{D}_n^{x_t}$. **client** sends $S' = \texttt{Resample}(S, x_t)$ to the $\mathbf{server}_2$.

By Property 1 defined in Section 3, S' is computationally indistinguishable from a fresh set sampled from $\mathbb{D}_n$. Therefore, we have that $\mathbf{Expt}_1^*$ and $\mathbf{Expt}_1$ are indistinguishable. Next, we define another intermediate experiment $\mathbf{Expt}_0^*$ to help in the proof.

– $\mathbf{Expt}_0^*$: Here, for each query index x_t that **client** receives, **client** interacts with $\mathbf{server}_2$ as in our PIR protocol, except that on the refresh phase after each query, instead of picking a table entry $B_k = (S_k, P_k)$ from our secondary sets and running $S'_k = \texttt{Add}(S_k, x_t)$, we generate a new random set $S \sim \mathbb{D}_n^{x_t}$ and replace our used set with sk instead.

First, we note that by Property 2 defined in Section 3, it follows directly that $\mathbf{Expt}_0$ and $\mathbf{Expt}_0^*$ are computationally indistinguishable. Now, we continue to show that $\mathbf{Expt}_0^*$ and $\mathbf{Expt}_1^*$ are computationally indistinguishable. At the beginning of the protocol, right after the offline phase, the client has a set of $|T|$ primary sets picked at random. For the first query index, x_1, we either pick an entry $(S_j, p_j) \in T$ from these random sets where $x_1 \in S_j$ or, if the that fails, we run $S_j \sim \mathbb{D}_n^{x_1}$.

Then, we send to $\mathbf{server}_2$ $S'_j = \texttt{Resample}(S_j, x)$. Note that the second case is trivially equivalent to generating a random set with x_1 and resampling it at x_1. But in the first case, note that T holds a sets sampled from $\mathbb{D}_n$ in order. As a matter of fact, looking at it in this way, S_j is the first output in a sequence of samplings that satisfies the constraint of x being in the set. Then, if we consider just the executions from 1 to j, this means that picking S_j is equivalent to sampling from $\mathbb{D}_n^{x_1}$, by definition. Then, by Property 1, it follows that the set that the server sees in the first query is indistinguishable from a freshly sampled set.

It follows from above that for the first query, q_1, $\mathbf{Expt}_0^*$ is indistinguishable from $\mathbf{Expt}_1^*$. To show that this holds for all q_t for $t \in [1, Q]$ we show, by induction, that after each query, we refresh our set table T to have the same distribution as initially. Then, by the same arguments above, it will follow that every query q_t in $\mathbf{Expt}_0^*$ is indistinguishable from each query in $\mathbf{Expt}_1^*$.

Base Case. Initially, our table T is a set of $|T|$ random sets sampled from $\mathbb{D}_n$ independently from the queries, offline.

Inductive Step. After each query q_t, the smallest table entry (S_j, p_j) such that $x_t \in S_j$ is replaced with a set sampled from $\mathbb{D}_n^{x_t}$. Since the sets are identically distributed, then it must be that the table of set keys T maintains the same distribution after each query refresh.

Since our set distribution is unchanged across all queries, then using the same argument as for the first query, each query q_t from **client** will be indistinguishable from a freshly sampled set to $\mathbf{server}_2$. Then, we can say that $\mathbf{Expt}_1^*$ is indistinguishable from $\mathbf{Expt}_0^*$. This concludes our proof for experiment indistinguishability. Since we have defined a way to simulate our protocol *without* access to each x_t, it follows that we satisfy $\mathbf{server}_2$ privacy for any PPT non-uniform adversary $\mathcal{A}$.

Correctness. To show correctness, we consider a slightly modified version of the scheme: After the refresh phase has used the auxiliary set (S_j, p_j), the client stores (S_j, p_j, z_j), where z_j is the element that was added to S_j as part of the protocol—for the sets that have not been used, we simply set $z_j = null$. Note that the rest of the

scheme functions exactly as in Figure 1 and therefore never uses z_j. It follows, then, that the correctness of this modified scheme is exactly equivalent to the correctness of the scheme we presented. Note that the query phase will fail to output the correct bit only on the following four occasions: *(Case 1).* x_i is not in any primary set that was preprocessed. *(Case 2).* The resampling does not remove x_i *(Case 3).* Resampling removes *more* that just x_i from the set. *(Case 4).* Parity is incorrect because `Add` added a related element during the refresh phase.

Case 1: From the privacy proof above, we know that refreshing the sets maintains the primary set distribution. Then, we can use the same argument as in Lemma 31 and say that, for a query x_i, for all $i \in \{1, \ldots, Q\}$, we have:

$$\Pr[x_i \notin \cup_{j\in[1,l]} S_j] = \left(\frac{1}{e}\right)^{\log n} \leq \frac{1}{n}\,.$$

Case 2: Since `Resample` is independent from the set (just tossing random coins), we can again re-use the proof of Lemma 31 and say that, for any x_i, for all $i \in \{1, .., Q\}$, we have:

$$\Pr[x_i \in \texttt{Resample}(S, x_i)] \leq \frac{1}{\sqrt{n}(\log n)^2}\,.$$

Case 3: Case 3 requires us to look into our modified scheme. For the initial primary sets, the probability of removing an element related to the query is exactly the same as in Case 3 for our Lemma 31. However, for sets that were refreshed, we need to consider the fact that these are not freshly sampled sets, in fact, they are sets that were sampled and then had an `Add` operation performed on them. For a given query x_i, let S_j be the first set in T that contains x_i. Let us denote `PuncRel` to be the event that we remove *more* than just x_i when resampling S_j on x_i. We split the probability of `PuncRel` as

$$\begin{aligned}\Pr[\texttt{PuncRel}] = \Pr[\texttt{PuncRel} \mid \texttt{Related}(x_i, z_j) = 1 \wedge x_i \neq z_j] \times \Pr[\texttt{Related}(x_i, z_j) = 1 \wedge x_i \neq z_j] \\ \cup \Pr[\texttt{PuncRel} \mid \texttt{Related}(x_i, z_j) = 0 \vee x_i = z_j] \times \Pr[\texttt{Related}(x_i, z_j) = 0 \vee x_i = z_j]\,.\end{aligned}$$

The first term corresponds to the case where the added element in a previous refresh phase, z_j, is related to the current query element, x_i. Note that if x_i equals z_j, we get the same distribution as the initial S_j by Property 2 in Section 3. Then, we consider only the case where z_j does not equal x_i. Note that we can bound

$$\Pr[\texttt{Related}(x_i, z_j) = 1 \wedge x_i \neq z_j] \leq \Pr[\texttt{Related}(S_j, z_j) = 1] \leq \frac{1}{2\log n}\,.$$

Above, we use $\texttt{Related}(S_j, z_j)$ to denote the probability that there is any related element to z_j (not equal to z_j) in S_j. We can bound this event by Lemma 31 (see Case 3). Then, we have

$$\Pr[\texttt{PuncRel} \mid \texttt{Related}(x_i, z_j) = 1 \wedge x_i \neq z_j] \times \Pr[\texttt{Related}(x_i, z_j) = 1 \wedge x_i \neq z_j] \leq \frac{1}{2\log n}\,.$$

For the second term of our initial equation, since $\texttt{Related}(x_i, z_j)$ is 0 or x_i equals z_j, note that our probability of resampling incorrectly is either *independent* of z_j, since z_j does not share any prefix with x_i and therefore the resampling cannot affect z_j or its related elements in any way, by definition; *or* it is identical to the probability of the initial set, by Property 2. Therefore, we have that the probability of removing a related element is at most the probability of removing a related element in the original set, which by Lemma 31 is

$$\Pr[\texttt{PuncRel} \mid \texttt{Related}(x_i, z_j) = 0 \vee x_i = z_j] \leq \frac{1}{2\log n}.$$

And, therefore, it follows that

$$\Pr[\texttt{PuncRel} \mid \texttt{Related}(x_i, z_j) = 0 \vee x_i = z_j] \times \Pr[\texttt{Related}(x_i, z_j) = 0 \vee x_i = z_j] \leq \frac{1}{2\log n}.$$

Finally, we have that $\Pr[\texttt{PuncRel}] \leq \frac{1}{2\log n} + \frac{1}{2\log n} \leq \frac{1}{\log n}$.
Case 4: Lastly, we have the case that query x_i is incorrect because the parity p_j from the set S_j where we found x_i is incorrect. This will only happen when we added elements related to z_j when adding z_j during the refresh phase. We denote this event $\texttt{AddRel}$. By Lemma B1, we have that

$$\Pr[\texttt{AddRel}] \leq \frac{1}{2\log n}.$$

We can conclude that at each query x_i, $i \in \{1, \ldots, Q\}$, assuming the previous query was correct, it follows that the probability of a query being incorrect, such that the output of the query does not equal $\mathsf{DB}[x_i]$, is:

$$\Pr[\text{incorrect query}] \leq \frac{1}{n} + \frac{1}{\sqrt{n}\log^2 n} + \frac{1}{\log n} + \frac{1}{2\log n} \leq \frac{2}{\log n} \leq \frac{1}{3} \text{ for } n > 405.$$

Because at each step we run a majority vote over $\omega(\log n)$ parallel instances, we can guarantee that, since our failure probability is less than $\frac{1}{2}$, each instance will get back the correct $\mathsf{DB}[x_i]$ with overwhelming probability. □

C PRS Contructions and Proofs

This section presents a construction and proof for the Adaptable PRS, as introduced and defined in Section 4. We present a construction of our Adaptable PRS in Figure 3. In the proof, we use a function $\texttt{time}$: $f(\cdot) \to \mathbb{N}$ that takes in a function $f(\cdot)$ and output the number of calls made in $f(\cdot)$ to any PRF function. We also prove Theorem 41 for our construction in Figure 3. We prove Thorem 41 below:

Proof. We begin the proof by showing that our scheme in Figure 3 satisfies the definitions in Appendix A. We then argue efficiencies.

Let $B = 2\log\log n$, $m = \frac{1}{2}\log n + B$.

• $\mathsf{Gen}(1^\lambda, n) \to (sk, msk)$:

1. Let $msk_0 \leftarrow \text{PRF.}\mathsf{Gen}(1^\lambda, \log n + B, m)$, $msk_1 \leftarrow \text{PRF.}\mathsf{Gen}(1^\lambda, \log n + B, m)$.
2. Let P_1, P_2 be two sets of random $\left(\frac{1}{2}\log n + B\right)$ strings in $\{0,1\}^{\log n + B}$ that start with a 1-bit.
3. Let $sk_0 = \text{PRF.}\mathsf{Puncture}(msk_0, P_1)$, $sk_1 = \text{PRF.}\mathsf{Puncture}(msk_1, P_2)$.
4. output $(sk, msk) = ((sk_0, sk_1), (msk_0, msk_1))$.

• $\mathsf{Eval}(sk, x) \to b$: % *internal function used to simplify algorithms*

1. Return $\text{PRF.}\mathsf{PEval}(sk[0], x) \oplus \text{PRF.}\mathsf{PEval}(sk[1], x)$.

• $\mathsf{EnumSet}(sk) \to S$:

1. Let $Z_{\frac{1}{2}\log n}$ be all bit-strings in $\ell \in \{0,1\}^{\frac{1}{2}\log n}$ such that $\mathsf{Eval}(sk, l) = 1$.
2. Then, For $i \in [\frac{1}{2}\log n + 1, \ldots, \log n]$:
   (a) Set Z_{i+1} to be any string of the form $b||\ell$ where $b \in \{0,1\}$, $\ell \in Z_i$ and $\mathsf{Eval}(sk, b||\ell) = 1$.
3. Return $S = \left\{\ell : \ell \in Z_{\log n} \wedge \mathsf{Eval}(sk, 0^k||\ell) = 1 \text{ for } k \in [0, B]\right\}$.

• $\mathsf{InSet}(sk, x) \to b$:

1. Let $z = 0^B||x$.
2. Output 1 if $\mathsf{Eval}(sk, z[i:]) = 1$ for $i \in [0, m]$, otherwise output 0.

• $\mathsf{Resample}(msk, sk, x) \to sk$:

1. Let $z = 0^B||x$, $Z = \{z[i:]\}$ for $i \in [0, m]$.
2. Let $sk_x = \text{PRF.}\mathsf{Puncture}(msk[1], Z)$.
3. Return $(sk[0], sk_x)$.

• $\mathsf{Add}(msk, sk, x) \to (msk, sk)$:

1. Write $x \in \{0,1\}^{\log n}$ as a binary string.
2. Define $z = 0^B||x$, $Z = \{z[i:]\}$ for $i \in [0, m]$.
3. While `true`: % *puncture until we find* sk_x *such that* $\mathsf{Eval}(sk, x)$ *equals* 1.
   (a) Let $sk_x = \text{PRF.}\mathsf{Puncture}(msk[0], Z)$.
   (b) If $\mathsf{InSet}((sk_x, sk[1]), x)$, break.
4. Output $((\textit{null}, msk[1]), (sk_x, sk[1]))$

Fig. 3. Our Adaptable PRS Implementation.

Correctness and pseudorandomness with respect to $\mathbb{D}_n$. Correctness follows from our construction and functionality preservation of the underlying PRF. Pseudorandomness follows from pseudorandomness of the underlying PRF (Definition E1). Both incur a negligible probability of failure in λ, inherited from the underlying PRF.
Functionality preservation in resampling and addition. Assuming pseudorandomness and functionality preservation of the underlying PRF (Definition E1 and Definition E2), our PRS scheme satisfies the properties of Functionality Preservation in Addition.

For $(sk, msk) \leftarrow \mathsf{Gen}(1^\lambda, n)$ until $\mathsf{InSet}(sk, x)$, and $sk_x \leftarrow \mathsf{Punc}(msk, sk, x)$:

- From construction, $\mathsf{EnumSet}(sk_x) \subseteq \mathsf{EnumSet}(sk)$, since puncturing strings that evaluate to 1 can only reduce the size of the set (since we only resample elements in the set).
- From the point above, and construction of our $\mathsf{EnumSet}$, it follows that $\texttt{time}(\mathsf{EnumSet}(sk)) \geq \texttt{time}(\mathsf{EnumSet}(sk_x))$.
- By construction of our resampling operation and `Related` function, it must be that
$$y \in \mathsf{EnumSet}(sk) \setminus \mathsf{EnumSet}(sk_x) \leftrightarrow \texttt{Related}(x, y) = 1.$$

Also, for any $n, \lambda \in \mathbf{N}$, $x \in \{0, \ldots, n-1\}$, for $(sk, msk) \leftarrow \mathsf{Gen}(1^\lambda, n)$, $sk^x \leftarrow \mathsf{Add}(msk, sk, x)$ we note that:

- By construction, $\mathsf{EnumSet}(sk) \subseteq \mathsf{EnumSet}(sk^x)$ since since we only ever make 0s into 1s.
- By the converse of same argument as Functionality Preservation in Resampling above, it follows that
$$y \in \mathsf{EnumSet}(sk^x) \setminus \mathsf{EnumSet}(sk) \leftrightarrow \texttt{Related}(x, y) = 1.$$

Therefore, our scheme satisfies Functionality preservation in resampling and addition.
Security in resampling. We show that our scheme satisfies Definition A2 below, assuming pseudorandomness and privacy w.r.t. puncturing of the underlying PRF (Definition E1 and Definition E3, respectively).

To aid in the proof, we define an intermediate experiment, $\mathbf{Expt}_1^*$, defined as:

- $\mathbf{Expt}_1^*$: Run $\mathsf{Gen}(\lambda, n) \to (sk, msk)$, and return $sk_x \leftarrow \mathsf{Resample}(msk, sk, x)$.

For each sk output by Gen, $sk = (sk[0], sk[1])$, two keys of m-puncturable PRFs. First, we show indistinguishability between $\mathbf{Expt}_1^*$ and $\mathbf{Expt}_0$:

Assume that there exists a distinguisher D_0 than can distinguish $\mathbf{Expt}_1^*$ and $\mathbf{Expt}_0$. Let us say that D_0 outputs 0 whenever it is on $\mathbf{Expt}_0$ and 1 when it is on $\mathbf{Expt}_1^*$. Then, we can construct a D_0^* with access to D_0 that breaks the privacy w.r.t. puncturing of the PRF (as in Definition E3) as follows, for any $x \in \{0, \ldots, n-1\}$:

Let $m = \frac{1}{2}\log n + B$, $L = \log n + B$, $z = 0^B||x$.
$D_0^*(m, L, z)$:

1. Define $P_0 = \{z[i:]\}_{i\in[0,m]}$ and let P_1, P_2 be a set of m random points of length L starting with a 1-bit.
2. Send P_0, P_1 to the privacy w.r.t. puncturing experiment and get back sk_{P_b} and oracle access to PRF.$\mathsf{Eval}(sk, \cdot)$.
3. Run PRF.$\mathsf{Gen}(1^\lambda, L, m) \to sk$, PRF.$\mathsf{Puncture}(sk, P_2) \to sk_{P_2}$.
4. Set secret key $sk' = (sk_{P_2}, sk_{P_b})$.
5. Return $D_0(sk')$.

Note that in the case where b equals 0, the experiment is exactly equivalent to D_0's view of $\mathbf{Expt}_0$, since sk' is two random m-privately-puncturable PRF keys punctured and m points starting with a 1-bit. Also, when b is 1, D_0's view is exactly equivalent to $\mathbf{Expt}_1^*$, since we pass in two random m-privately-puncturable PRF keys, one punctured at m points starting with a 1-bit, and the other at $\{z[i:]\}_{i\in[0,m]}$, with no constraints on whether x was in the set before or after the puncturings. Then, since D_0's view is exactly the same as its experiment, it will distinguish between both with non-negligible probability, and whatever it outputs, by construction, will be the correct guess for b with non-negligible probability.

Now we proceed to show that $\mathbf{Expt}_1^*$ and $\mathbf{Expt}_1$ are indistinguishable, assuming pseudorandomness of the underlying PRF. Now, assume there exists a distinguisher D_1 that can distinguish between $\mathbf{Expt}_1^*$ and $\mathbf{Expt}_1$ with non-negligible probability. Then, we can construct a distinguisher D_1^* that uses D_1 to break the pseudorandomness of the underlying PRF (as in Definition E1) as follows, for any $x \in \{0, \ldots, n-1\}$:

Let $m = \frac{1}{2}\log n + B$, $L = \log n + B$, $z = 0^B||x$.
$D_1^*(m, L, z)$:

1. Send $P = \{z[i:]\}_{i\in[0,m]}$ to the PRF pseudorandomness experiment, get back sk_P and a set of m bits $\{M_i\}_{i\in[0,m]}$.
2. Let P_1 be a set of m random bit strings of length L starting with a 1-bit. Run PRF.$\mathsf{Gen}(1^\lambda, L, m) \to sk$, PRF.$\mathsf{Puncture}(sk, P_1) \to sk_{P_1}$. Let $sk' = (sk_{P_1}, sk_P)$.
3. If $\forall i \in [0, m]$, PRF.$\mathsf{PEval}(sk_{P_1}, z[i:]) \oplus M_i = 1$, output $D_1(sk')$, else output a random bit.

Note that in the case D_1's view in the case where the evaluations as described above all output 1 is exactly its view in distinguishing between our $\mathbf{Expt}_1$ and $\mathbf{Expt}_1^*$. With probability $\frac{1}{2}$, it is given a punctured key where x was an element of the original set, and with probability $\frac{1}{2}$ it is given a punctured key where x was sampled at random. Then, in this case, it will be able to distinguish between the two with non-negligible by assumption, and therefore distinguish between the real and random experiment for pseudorandomness of the PRF. Since the probability of having all the evaluations output 1 is non-negligible, then we break the pseudorandomness of the PRF. By contraposition, then, assuming pseudorandomness of the PRF, it must be that $\mathbf{Expt}_1$ and $\mathbf{Expt}_1^*$ are indistinguishable. This concludes our proof.

Security in addition. We now show that our scheme satisfies Definition A4, assuming privacy w.r.t. puncturing of the underlying PRF (Definition E3). Assume there exists a

distinguisher D that can distinguish between these two with non-negligible probability. Then, we can construct a distinguisher D^* that breaks privacy w.r.t. puncturing of the PRF as follows, for any $x \in \{0, \ldots, n-1\}$:

Let $m = \frac{1}{2}\log n + B$, $L = \log n + B$, $z = 0^B || x$.
$D^*(m, L, z)$:

1. Define $P_0 = \{z[i:]\}_{i \in [0,m]}$ and let P_1, P_2 be two sets of random m points of length L starting with a 1-bit.
2. Send P_0, P_1, to the privacy w.r.t. puncturing experiment and get back sk_{P_b} and oracle access to PRF.$\mathsf{Eval}(sk, \cdot)$.
3. Run PRF.$\mathsf{Gen}(1^\lambda, L, m) \rightarrow (msk, sk)$, PRF.$\mathsf{Puncture}(sk, P_2) \rightarrow sk_{P_2}$.
4. Set our secret key $sk' = (sk_{P_b}, sk_{P_2})$.
5. **If** $\mathsf{InSet}(sk', x)$, output $D(sk')$, **else** output a random bit.

Consider the case where $x \in \mathsf{EnumSet}(sk')$:

- If P_0 was punctured, D's view is exactly equivalent to $\mathbf{Expt}_0$ in his experiment, since in Add we output a secret key $sk = (sk[0], sk[1])$ where $sk[0]$ is punctured at x, $sk[1]$ is punctured at m random points starting with a 1, and $\mathsf{InSet}(sk, x)$ returns true.
- If P_1 was punctured, D's view is exactly equivalent to $\mathbf{Expt}_1$ in his experiment, by construction of Gen, P_1 and P_2, the sk outputted is equivalent to a key outputted by $\mathsf{Gen}(1^\lambda, n)$ where $\mathsf{InSet}(sk, x)$ returns true.

We conclude that, conditioned on $\mathsf{InSet}(sk_{P_b}, x)$ returning true, D's view of the experiment is exactly equivalent to the experiment from our Definition A4, and therefore it will be able to distinguish between whether P_0 and P_1 was punctured with non-negligible probability. If we fix a random $sk[1]$, the probability:

$$\Pr\left[\mathsf{InSet}(sk', x) = true\right] = \frac{1}{\sqrt{n}} > \mathtt{negl}(n).$$

Then, the algorithm D^* we constructed will break the privacy w.r.t. puncturing of the PRF with non-negligible probability. By contraposition, assuming privacy w.r.t. puncturing, sk^x and sk are computationally indistinguishable. Following almost exactly the same argument as above, we can show that the tuples $(sk^x[0], msk^x[1])$ and $(sk[0], msk[1])$ are also indistinguishable. Also, in both tuples $(msk^x[1], sk^x[1])$ and $(msk[1], sk[1])$ the master key is just the unpunctured counterpart of the secret key. Finally, $msk^x[0] = msk[0] = null$. Then, since we have shown that assuming the privacy w.r.t. puncturing property, the keys involved are pairwise indistinguishable, by the transitive property, we see that assuming privacy w.r.t. puncturing, (msk^x, sk^x) and (msk, sk) are computationally indistinguishable and therefore, *security in addition* holds.

Efficiencies. Efficiency for our Gen,InSet and $\mathsf{Resample}$ follow from the construction and efficiencies for our underlying PRF. The two efficiencies which we will show are $\mathsf{EnumSet}$ and Add.

Note that in $\mathsf{EnumSet}$, the step 1 takes $\widetilde{O}(\sqrt{n})$ time to evaluate every string of size $\frac{\log n}{2}$, then, by pseudorandomness of the PRF, at each subsequent step we only ever keep

$\sqrt{n}$ strings since half are eliminated. Since there are a logarithmic number of steps, we can say that $\mathsf{EnumSet}$ runs in probabilistic $\widetilde{O}(\sqrt{n})$ time.

For Add, by pseudorandomness of the PRF, our construction will take probabilistic $\widetilde{O}(\sqrt{n})$ time. (We can show that we can bound our scheme to have deterministic $\widetilde{O}(\sqrt{n})$ Add and $\mathsf{EnumSet}$ by incurring some small error probability that is dealt with through the parallel instances. We can also reduce the time of Add to $\widetilde{O}(1)$ by exploring the underlying PRF construction. This is not central to our approach, so it we provided it as supplementary material Appendix E.2 to the interested reader.) □

D Constructions and Proofs for Section 5

In this section, we give proof for our scheme presented in Figure 2. Then, we give a full construction and proof for the 1PIR scheme from Theorem 52.

D.1 Proving Theorem 51

We introduce a small lemma that will aid us in our task. This lemma, intuitively, tells us that our APRS's $\mathsf{Resample}$ is exactly equivalent to our `Resample` operation defined in Section 3 (incurring some negligible probability of failure), in other words, the new evaluations of punctured points are pseudorandom and completely independent of the previous evaluations before puncturing.

Lemma D1 (Randomness in resampling)**.** *In some distribution $\mathbb{D}_n$, the following two distributions are computationally indistinguishable for $m = \frac{1}{2}\log n + B$ and any $x \in \{0, \ldots, n-1\}$:*

- **Expt**$_0$: Run $\mathsf{Gen}(1^\lambda, n) \to (sk, msk)$ until $\mathsf{InSet}(sk, x)$, run $sk_x \leftarrow \mathsf{Resample}(msk, sk, x)$, Return the tuple $(\mathsf{EnumSet}(sk), x \in \mathsf{EnumSet}(sk_x))$.
- **Expt**$_1$: Run $\mathsf{Gen}(1^\lambda, n) \to (sk, msk)$ until $\mathsf{InSet}(sk, x)$, sample some boolean b from Bernoulli(ϕ) where $\phi = 2^{-m}$. Output the tuple $(\mathsf{EnumSet}(sk), \text{Bernoulli}(\phi))$.

Note that $x \in \mathsf{EnumSet}(sk)$ denotes a boolean denoting true or false for the expression.

Proof. Let us define an intermediary experiment to aid in the proof.

- **Expt**$_0^*$: Run $(sk, msk) \leftarrow \mathsf{Gen}(1^\lambda, n)$ once and let $sk^x \leftarrow \mathsf{Add}(msk, sk, x)$. Return the tuple $(\mathsf{EnumSet}(sk^x), x \in \mathsf{EnumSet}(sk))$.

Note that by our two security properties, it follows that **Expt**$_0$ and **Expt**$_0^*$ are indistinguishable, since generating a key until finding one with x is equivalent to adding, and sampling a key with x and resampling x is indistinguishable from sampling a fresh key. Then, by pseudorandomness of the PRF, it follows that $x \in \mathsf{EnumSet}(sk)$ for a fresh sk is indistinguishable from Bernoulli(ϕ) expect with negligible probability, and by security in addition as we saw above generating a set key until we find one with x is indistinguishable from generating a fresh key and adding x. Then, it follows that **Expt**$_1$ and **Expt**$_0^*$ are also indistinguishable, and this concludes our proof.

Now, we have all the tools we need to prove Theorem 51. See the proof below:

Theorem 51 (Efficient 2PIR+ protocol). *Assuming* LWE*, the 2PIR+ scheme in Figure 2 is correct (per Definition 22) and private (per Definition 23) and has: (i)* $\widetilde{O}(\sqrt{n})$ *client storage and* $\widetilde{O}(\sqrt{n})$ *client time; (ii)* $\widetilde{O}(\sqrt{n})$ *amortized server time and no additional server storage; (iii)* $\widetilde{O}(1)$ *amortized bandwidth.*

Proof. We use an APRS scheme $APRS = (\mathsf{Gen}, \mathsf{EnumSet}, \mathsf{InSet}, \mathsf{Resample}, \mathsf{Add})$ that satisfies Theorem 41. Efficiencies follow from the protocol and from Theorem 41.
Privacy. Privacy for $\mathbf{server}_1$ is trivial. It only ever sees random sets generated completely independent of the queries and is not interacted with online. We present the privacy proof for $\mathbf{server}_2$ below.

Privacy with respect to $\mathbf{server}_2$, as per our definition, must be argued by showing there exists a stateful algorithm $\mathtt{Sim}$ that can run without knowledge of the query and be indistinguishable from an honest execution of the protocol, from the view of any PPT adversary $\mathcal{A}$ acting as $\mathbf{server}_2$ for any protocol $\mathbf{server}_1^*$ acting as $\mathbf{server}_1$. First, we note that the execution of the protocol between **client** and $\mathbf{server}_2$ is independent of **client**'s interaction with $\mathbf{server}_1$. **client** generates sets and queries $\mathbf{server}_1$ in the offline phase for their parity. Although this affects correctness of each query, it does not affect the message sent to $\mathbf{server}_2$ at each step of the online phase, since this is decided by the sets, generated by **client**. Then, we can rewrite our security definition, equivalently, disregarding **client**'s interactions with $\mathbf{server}_1$.

We want to show that for any query q_t for $t \in [1, Q]$, q_t leaks no information about the query index x_t to $\mathbf{server}_2$, or that interactions between **client** and $\mathbf{server}_2$ can be simulated with no knowledge of x_t. To do this, we show, equivalently, that the following two experiments are computationally indistinguishable. (Irrespective of the query index x_t, therefore, encompassing the adaptiveness required by the definition.)

- $\mathbf{Expt}_0$: Here, for each query index x_t that **client** receives, **client** interacts with $\mathbf{server}_2$ as in our PIR protocol.
- $\mathbf{Expt}_1$ In this experiment, for each query index x_t that **client** receives, **client** *ignores* x_t, samples a fresh APRS key $(sk, msk) \leftarrow \mathsf{Gen}(1^\lambda, n)$ and sends sk to $\mathbf{server}_2$.

First we define an intermediate experiment $\mathbf{Expt}_1^*$.

- $\mathbf{Expt}_1^*$: For each query index x_t that **client** receives, **client** samples $(sk, msk) \leftarrow \mathsf{Gen}(1^\lambda, n)$ *until* $\mathsf{InSet}(sk, x_t) = true$ and sends $sk' = \mathsf{Resample}(msk, sk, x_t)$ to the $\mathbf{server}_2$.

Note that from our Security in Resampling property (Definition A2), it follows directly that $\mathbf{Expt}_1^*$ and $\mathbf{Expt}_1$ are computationally indistinguishable. Next, we define another intermediate experiment $\mathbf{Expt}_0^*$ to help in the proof.

- $\mathbf{Expt}_0^*$: Here, for each query index x_t that **client** receives, **client** interacts with $\mathbf{server}_2$ as in our PIR protocol, except for every set key tuple in T, $msk[0]$, is set to $null$ offline. Upon generating new sets (when x_t is not found), we also set $msk[0]$ to $null$.

It is not hard to see that setting $msk[0]$ to $null$ for each set key tuple in T does not change the **server**$_2$'s view, since $msk[0]$ is never used or seen by **server**$_2$. Similarly, when generating a set (for the case where x_t) is not found, and after Add for the refresh, $msk[0]$ is not used, and therefore setting $msk[0]$ to $null$ in these cases is a local change that cannot affect **server**$_2$'s view of the queries. We therefore conclude that **Expt**$_0$ and **Expt**$_0^*$ are statistically (and computationally) indistinguishable.

Next, we define our last hybrid experiment in our proof:

– **Hyb:** This experiment runs exactly like **Expt**$_0^*$, except that on the refresh phase after each query, instead of picking a table entry $B_k = ((msk_k, sk_k), p_k)$ from our secondary sets, running $sk' = \mathsf{Add}(msk_k, sk_k, x_t)$, and replacing our set with $((msk_k, sk'), p_k)$, we generate a new random set $(msk, sk) \leftarrow \mathsf{Gen}(1^\lambda, n)$ until $\mathsf{InSet}(msk, sk, x_t)$ is $true$, set $msk[0] = null$, and replace our used set-parity tuple with $((msk, sk), 0)$ instead.

Note that differentiating between **Expt**$_0^*$ and **Hyb** is exactly breaking the Security in Addition experiment (Definition A4). Then, by the security in addition, it follows that **Expt**$_0^*$ and **Hyb** are computationally indistinguishable from **server**$_2$'s view.

Now, finally, must show that **Hyb** and **Expt**$_1^*$ are computationally indistinguishable. At the beginning of the protocol, on **Hyb**, right after the offline phase, the client has a set of $|T|$ (msk, sk) pairs, where $msk_i[1]$ was set to $null$ for every $i \in \{1, \ldots, |T|\}$. For the first query index, x_1, we either pick an entry $((msk_j, sk_j), p_j) \in T$ from these random sets where $\mathsf{InSet}(sk_j, x_1) = true$ or, if the that fails, we set $j = |T| + 1$ and run $(msk_j, sk_j) \leftarrow \mathsf{Gen}(1^\lambda, n)$ until $\mathsf{InSet}(sk, x_1)$. (Again, here we set $msk_j[1]$ to be $null$ as per **Hyb**'s experiment definition.)

Then, we send to **server**$_2$ $sk' = \mathsf{Resample}(msk_j, sk_j, x_1)$. Note that the second case is trivially equivalent to generating a random set with x_1 and resampling it at x_1. But in the first case, note that T holds a set of keys sampled with Gen in order. As a matter of fact, looking at it in this way, sk_j is the first output in a sequence of samplings that satisfies the constraint of x_1 being in the set. Then, if we consider just the executions from 1 to j, this means that picking sk_j is equivalent to running $(msk_j, sk_j) \leftarrow \mathsf{Gen}(1^\lambda, n)$ until we find sk_j such that $\mathsf{InSet}(sk_j, x_1) = true$, by definition. Then, by Security in Resampling (Definition A2), it follows that the set that the server sees in the first query is indistinguishable from a freshly sampled set.

It follows from above that for the first query, q_1, **Hyb** is indistinguishable from **Expt**$_1^*$. To show that this holds for all q_t for $t \in [1, Q]$ we show, by induction, that after each query, we refresh our set table T to have a distribution that is computationally indistinguishable from the distribution at the previous step. Then, by the same arguments above, it will follow that every query q_t in **Hyb** is computationally indistinguishable from each **Expt**$_1^*$ for every $t \in [1, Q]$. After Q queries, we re-run the offline phase with independent randomness, so if we can show the inductive step, our proof holds for unlimited queries. As previously, we will disregard the parities since they are exclusively used locally.

Base Case. Initially, our table T is a set of $|T|$ (msk, sk) pairs generated offline, independently from the queries, with $\mathsf{Gen}(1^\lambda, n)$ (and then setting $msk[0] = null$).

Inductive Step. After each query q_t, the table entry $((msk_j, sk_j))$ for the smallest j in $|T|$ such that $\mathsf{InSet}(sk_j, x_t) = true$ is replaced with a set generated as:

$(msk, sk) \leftarrow \mathsf{Gen}(1^\lambda, n)$ *until* $\mathsf{InSet}(sk, x_t) = true$ (and again, as per **Hyb**'s definition, setting $msk[0] = null$). Notice that these sets are generated in exactly the same fashion. We can view the offline generation of sets 1 to j as sampling until we generate a set that contains x_t, which means that the keys (msk_j, sk_j) generated offline and the new keys $(msk_s k)$ are generated in exactly the same fashion, and therefore statistically (and computationally) indistinguishable. Then it must be that the table of set keys T maintains the same distribution after we replace (msk_j, sk_j) by the new (msk, sk).

We have shown the inductive step and therefore conclude that the distribution of T is computationally indistinguishable across Q queries. Then, we can say that $\mathbf{Expt}_1^*$ is computationally indistinguishable from **Hyb**. This concludes our proof for experiment indistinguishability. Since we have defined a protocol *without* access to each x_t (picked adaptively), and shown through a series of hybrid experiements that it is computationally indistinguishable from running our real protocol defined in Figure 2 from the $\mathbf{server}_2$'s view, it follows that we satisfy privacy for any PPT non-uniform adversary $\mathcal{A}$ (as per Definition 23). We note that correctness follows very similarly from the proof of Theorem 31, except our correctness loses a negligible factor of correctness from our APRSs being computationally correct (inherited from the underlying ppPRF), and we use the Randomness in Resampling property to show that our Resample maps exactly as the computational counterpart of the `Resample` presented in Section 3. □

D.2 Theorem 52

As mentioned in Section 5, in order to port our two-server PIR scheme in Figure 2 to single server, we require three building blocks:

- *Batch Parity Circuit.* We will use a batch parity boolean circuit C. Given any database of size n and l lists of size m, C computes the parity of the l lists in $\widetilde{O}(l * m + n)$ time. A construction for such C was idealized and proved in [15] [5]. We used a slightly modified verysion of C, described below.
- *A Circuit for EnumSet.* Our EnumSet evaluation is in the RAM model. We must show that it can be converted to the circuit model maintaining its time complexity, so that it can be evaluated under FHE.
- *Gate-by-gate FHE.* We require the existence of a symmetric key FHE scheme (Gen,Enc,Dec,Eval) that is *gate-by-gate* (as defined in [15]), where *gate-by-gate* means Eval runs in time $\widetilde{O}(|C|)$ for a circuit of size $|C|$. As noted in [15], this is a property of standard FHE schemes [9, 26].

To use this for our new PIR algorithm we require two adjustments:

1. We modify the EnumSet algorithm in Figure 3 to keep arrays instead of sets. The first array is a tuple of all ℓ and their evaluation under sk. Subsequently, we define Z_{i+1} keep the tuples of the string $b||\ell$ and the multiplication between $\mathsf{Eval}(sk, b||\ell)$ and the previous evaluation present in Z_i, for all $\ell \in \mathsf{Z}_i$. Finally, we obliviously sort Z_{i+1} by the evaluation bit and keep only the first $\sqrt{n}log^2(n)$ elements in the sorted array. Note that:

[5] This circuit does not have polylog depth and therefore we require a circular-security assumption on FHE.

(a) At each step, Z_i will have $\sqrt{n}\log^2(n)$ elements, so oblivious sorting runs in $O(\sqrt{n}\log^2(n))$ time for each i and so the runtime of EnumSet is still $\widetilde{O}(\sqrt{n})$. The elements with evaluation '1' will move to Z_{i+1} exactly as in the original algorithm.
(b) In the end, we output the array (with evaluation bits) and denote the element to be in the set only if its final evaluation bit is 1. Note that while we will be able to fully express shorter sets, some large sets might have incorrect memberships due to elements in the set being chopped off at each step. By a Markov bound, we will show that the probability that this happens is small ($< 1/\log(n)$) and then deal with this through the parallel instantiations.

2. We modify the circuit by Corrigan-Gibbs et al. to enumerate our set through this array of size $n\log^2(n)$ and also multiply the database element by the evaluation bit, to ensure that elements not in the set in the final array (with an evaluation bit of 0) do not affect the parity.

With this modified PRS enumeration algorithm and circuit C, along with the FHE scheme introduced in 5 we finally construct our single server PIR scheme in Figure 4. The EnumSet algorithm run under FHE in Figure 4 uses these modifications to run in $\widetilde{O}(\sqrt{n})$ time.

Online, we know that keys that enumerate sets larger than $\sqrt{n}\log^2 n$ will have incorrect queries. However, Lemma D2 bounds the probability of this happening to be small.

Lemma D2 (Set Size Bound)**.** *Let* $\texttt{LargeSet}(\cdot)$ *be a function that takes in a key* sk *and outputs* 1 *if for any* $i \in \{0, \ldots, \log n/2\}$ *the set of candidate elements* Z_i *for the set defined by* EnumSet(sk) *contains more than* $\sqrt{n}\log^2 n$ *elements, and* 0 *otherwise. Then,*

$$\Pr\left[\texttt{LargeSet}\left(sk\right)\right] < \frac{1}{\log n}.$$

Proof. At the initial step, we have that the expected number of elements in our list is $\sqrt{n}/2$ (there are $\sqrt{n}$ bitstrings of size $\log n/2$. Furthermore, for each i, the expected number of elements that we keep is also $\sqrt{n}/2$, since we double the amount of strings in the previous step, but in expectation only keep half, by pseudorandomness of the PRF. Then, by a simple Markov bound we get that, for each intermediate set of good evaluations, Z_i, for some given i:

$$\Pr[|Z_i| > \sqrt{n}\log^2(n)] < \frac{1}{\log^2 n}.$$

Then, by a simple union bound of $\log n/2$ steps, we get that the probability that any of these is larger that $\sqrt{n}\log^2 n$ is less than $\frac{1}{\log n}$. □

From this Lemma, we see that this restriction incurs an additional correctness failure of $1/\log n$ compared to our normal scheme. This bound is not tight, but enough for our purposes.

Below, we prove that our scheme satisfies Theorem 52:

Run $v = \omega(\log \lambda)$ instances of the following scheme. Let $\ell = \sqrt{n}(\log n)^3$, $s = \ell + \sqrt{n}$.

Offline phase

- **client** generates s PRS keys $(msk_1, sk_1), \ldots, (msk_s, sk_s)$ each with $\mathsf{Gen}(1^\lambda, n)$.
- **client** encrypts all the secret keys, FHE.$\mathsf{Enc}(sk_1), \ldots,$ FHE.$\mathsf{Enc}(sk_s) \rightarrow (esk_1, \ldots, esk_s)$ and sends these to **server**$_1$.
- **server**$_1$ runs FHE.$\mathsf{Eval}(\mathsf{EnumSet}(esk_i))$ on each $esk_i, i \in [1, s]$ and gets back s sets $S_1, \ldots, S_s$, where it will be clear which $S_i = \perp$ from the size.
- **server**$_1$ evaluates the parity of each set under FHE using C, computes $ep_1, \ldots, ep_s$, and sends these to **client**.
- **client** decrypts each ep_i using FHE.Dec into the parity p_i and stores the first ℓ hints in $T = \{T_j = ((msk_j, sk_j), p_j)\}_{j \in [1,\ell]}$, and the next $\sqrt{n}$ hints in $B = \{B_k = ((msk_k, sk_k), p_k)\}_{k \in [\ell+1, \ell+\sqrt{n}]}$.

Online Phase: Invoked with some index $x \in \{0, .., n-1\}$.

• Query

1. **client** finds smallest j s.t. $T_j = ((msk_j, sk_j), p_j) \in T$ and $\mathsf{InSet}(sk_j, x)$. If no such j is found, we let $j = |T| + 1$, run $\mathsf{Gen}(1^\lambda, n, x) \rightarrow (sk_j, msk_j)$, let p_j be a uniform random bit.
2. **client** sends $sk' = \mathsf{Resample}(msk_j, sk_j, x)$ to **server**$_1$, that returns $r = \oplus_{k \in \mathsf{EnumSet(sk')}} \mathsf{DB}[k]$.
3. **client** computes $\mathsf{DB}[x] = r \oplus p_j$.
4. **client** computes $\mathsf{DB}[x]'$ to be the majority vote of the computed $\mathsf{DB}[x]$ over the v instances.

• Refresh (only run if $j \leq |T|$)

1. **client** gets $B_k = ((msk_k, sk_k), p_k)$ be the first item from set B.
2. **client** computes $(msk_k^x, sk_k^x) = \mathsf{Add}(msk_k, sk_k, x)$.
3. **client** sets $T_j = ((msk_k^x, sk_k^x), p_k \oplus (\mathsf{DB}[x]' \wedge \mathsf{InSet}(sk_k, x)))$, where T_j was the entry consumed by the query earlier, and also sets $B = B \setminus B_k$.

Fig. 4. Our 1PIR protocol using an Adaptable PRSet ($\mathsf{Gen}, \mathsf{EnumSet}, \mathsf{InSet}, \mathsf{Resample}, \mathsf{Add}$).

Proof. The efficiencies follow from the efficiencies in the scheme in Figure 2, except for extra polylogarithmic factors and λ factors incurred by using C and FHE in the offline phase, along with the extra number of preprocessed sets. Neither of these affect the complexity of our scheme when examined under $\widetilde{O}(\cdot)$. Note that $\mathsf{EnumSet}$ under FHE will still run in $\widetilde{O}(\sqrt{n})$ time using the modified algorithm described previously.

Privacy for the scheme follows from the security of the FHE scheme and privacy of the scheme in Figure 2 (Theorem 51).

Correctness follows from the correctness proof from Theorem 51 and Lemma D2, along with correctness of C and the FHE scheme we use. Note that for each single copy scheme, we incur exactly the same errors as in the 2PIR scheme, with the addition of the of an extra small error probability (less than or equal to $\frac{1}{\log n}$ at the query step). It is clear to see that this extra factor does not take the correctness probability of the single copy scheme to be greater than $\frac{1}{2}$ and therefore the same arguments from Theorem 51 hold. □

E Auxiliary Material

E.1 Previous Definitions for Privately-puncturable PRFs

We re-state here the definitions of security for privately-puncturable PRFs from previous works [7, 10, 36].

Definition E1 (Pseudorandomness for privately-puncturable PRFs)**.** *A privately-puncturable PRF scheme (*Gen*,* Eval*,* $\mathsf{Puncture}$*,* PEval*) satisfies* pseudorandomness *if no PPT admissible adversary $\mathcal{A}$ can distinguish between the following experiments (An adversary is admissible if it never queries elements in P on the original sk, and always picks a set P of size m.)*

- $\mathsf{Gen}(\lambda, L, m) \to sk$, $\mathcal{A}(\lambda) \to P$, $\mathsf{Puncture}(sk, P) \to sk_P$; $\mathcal{A}^{\mathsf{Eval}(sk,\cdot)}$ *is given* $(sk_P, \{\mathsf{Eval}(sk, x)\}_{x \in P})$.
- $\mathsf{Gen}(\lambda, L, m) \to sk$, $\mathcal{A}(\lambda) \to P$, $\mathsf{Puncture}(sk, P) \to sk_P$; $\mathcal{A}^{\mathsf{Eval}(sk,\cdot)}$ *is given* $(sk_P, \{R_i\}_{i=0,\ldots,m-1})$*, where R_i are sampled uniformly at random.*

Definition E2 (Functionality preservation in puncturing for privately-puncturable PRFs)**.** *A privately-puncturable PRF scheme (*Gen*,* Eval*,* $\mathsf{Puncture}$*,* PEval*) satisfies* functionality preservation *if for any PPT adversary $\mathcal{A}$ that outputs set of m points P of length $\leq L$ each, there exists a negligible function* $\mathtt{negl}(\cdot)$ *such that, for the following experiment:*

- $P \leftarrow \mathcal{A}(1^\lambda)$, $sk \leftarrow \mathsf{Gen}(1^\lambda, L, m)$, $sk_P \leftarrow \mathsf{Puncture}(sk, P)$.
- $x \leftarrow \mathcal{A}^{\mathsf{Eval}(sk,\cdot)}(sk_P)$.

it holds that

$$\Pr[(x \notin P) \wedge (\mathsf{Eval}(sk, x) \neq \mathsf{PEval}(sk_P, x))] \leq \mathtt{negl}(\lambda)\,.$$

Definition E3 (Privacy with respect to puncturing for privately-puncturable PRFs)**.** *A privately-puncturable PRF scheme (*Gen*,* Eval*,* Puncture*,* PEval*) satisfies* privacy with respect to puncturing *if for any PPT admissible adversary $\mathcal{A}$, experiments* $\text{Expt}^0(\lambda, L, m)$ *and* $\text{Expt}^1(\lambda, L, m)$ *are computationally indistinguishable. (An adversary is admissible if it never queries the oracle elements in $P_1 \cup P_2$ on the original sk, and always picks sets of size m.) Experiment* $\text{Expt}^b(\lambda, L, m)$ *is defined as follows.*

- $\mathsf{Gen}(\lambda, L, m) \to sk$, $\mathcal{A}(\lambda) \to (P_0, P_1)$, $\mathsf{Puncture}(sk, P_b) \to sk_{P_b}$.
- $b' \leftarrow \mathcal{A}^{\mathsf{Eval}(sk,\cdot)}(sk_{P_b})$.

E.2 Deterministic Time Bounds

We discuss below how get deterministic time bounds for our randomized algorithms used, EnumSet and Add.

EnumSet. To get deterministic run-time for EnumSet, we can cap the server enumeration time to be at most $6\sqrt{n}(\log n)^3$ function calls, after which it can output a random bit as the set parity. From a standard Markov argument, we see that this incurs an additional $\frac{1}{\log n}$ error per copy, which will be handled by the Chernoff bound. It is clear to see that this does not affect privacy for the servers, and only slightly affects correctness in a way that still leaves it overall greater than $\frac{1}{2}$ for any relevant n.

Add. To get deterministic run-time for Add, we can cap the execution similarly as above, with $2(\log n)^2$ iterations. This paired with Corollary 1 implies a deterministic time of $\widetilde{O}(1)$ for Add. We note that in order for this change not to affect privacy of the scheme, we must take precautions to change the order of steps in our PIR scheme to ensure privacy. To make this change, we must run step 1 and 2 of the Refresh phase along with step 1 of the Query phase. If either of these fails to execute correctly, then we send to the server $(sk, \_) \leftarrow \mathsf{Gen}(1^\lambda, n)$ and **client** sets $\mathsf{DB}[x]$ to be a uniform random bit. This introduces a small probability of correctness failure but does not affect privacy since we send a random set to $\textbf{server}_2$. If we do not take this precaution of running the steps of the Refresh phase upfront, then our PIR scheme for concrete performance bounds would potentially breach privacy in the case that Add fails.

In Corollary 1, we present a proof that we can run our Add function in expected $\widetilde{O}(1)$ time, as alluded to in the above and in Appendix C. Although this is not necessary for the efficiencies claimed in Theorem 41, it shows a significant improvement to the Add function; we note, however, that this speed-up only applies to the privately-puncturable PRF construction from Boneh et al. [7], where the $m-$privately-puncturable PRF is contructed from m $1-$privately-puncturable PRF keys. The final evaluation is the evaluation of the xor of all m keys, but the puncture operation punctures once at each key.

Corollary 1 (Efficient Add)**.** *Our construction can use the privately-puncturable PRF primitive to run Add run in $\widetilde{O}(1)$ time.*

Proof. We can make Add more efficient, from $\widetilde{O}(\sqrt{n})$ time to $\widetilde{O}(1)$ by using the construction above. For each point to be punctured, we puncture the corresponding key and individually check if it is mapped to 1, rather than attempting to get all punctures right at

once. Let us define PRF^1 to be the interface with the single-point privately-puncturable PRF (rather than the m-privately-puncturable PRF we used throughout the work). We replace the step 2 in the protocol with the steps as follows:

1. Write $msk[0]$ as $\{msk[0]_p\}_{p\in[1,m]}$.
2. Write $sk[0]$ as $\{sk[0]_p\}_{p\in[0,m]}$.
3. **For** i in $[0,m]$:
   (a) $p = \mathsf{Eval}(sk, z[i:])$.
   (b) $p_{i,old} = \text{PRF}^1.\mathsf{PEval}(sk[0]_i, z[i:])$.
   (c) $sk'_i = \text{PRF}^1.\mathsf{Puncture}(msk[0]_i, z[i:])$.
   (d) $p_{i,new} = \text{PRF}^1.\mathsf{PEval}(sk'_i, z[i:])$.
   (e) **If** $p \oplus p_{i,old} \oplus p_{i,new} \neq 1$ return to (c).
4. Let $sk' = \{sk'_i\}_{i\in[0,\frac{1}{2}\log n+B]}$.

It follows in a straightforward manner from construction of the PRF punctured at multiple points that our algorithm for addition presented earlier and the one using this technique output indistinguishable keys. This changes Add's run-time from expected $\widetilde{O}(\sqrt{n})$ to expected run-time $\widetilde{O}(1)$.