Private Information Retrieval with Sublinear Online Time

Henry Corrigan-Gibbs[1,2,3] and Dmitry Kogan[1]

[1] Stanford University, Stanford, CA, USA
[2] EFPL, Lausanne, Switzerland
[3] MIT CSAIL, Cambridge, MA, USA
henrycg@csail.mit.edu, dkogan@cs.stanford.edu

February 19, 2020

Abstract. We present the first protocols for private information retrieval that allow fast (sublinear-time) database lookups without increasing the server-side storage requirements. To achieve these efficiency goals, our protocols work in an offline/online model. In an *offline* phase, which takes place before the client has decided which database bit it wants to read, the client fetches a short string from the servers. In a subsequent *online* phase, the client can privately retrieve its desired bit of the database by making a second query to the servers. By pushing the bulk of the server-side computation into the offline phase (which is independent of the client's query), our protocols allow the online phase to complete very quickly—in time sublinear in the size of the database. Our protocols can provide statistical security in the two-server setting and computational security in the single-server setting. Finally, we prove that, in this model, our protocols are optimal in terms of the trade-off they achieve between communication and running time.

1 Introduction

A private information retrieval protocol [CGKS95, CGKS98] takes place between a client, holding an index $i \in [n]$, and a database server, holding a string $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$. The protocol allows the client to fetch its desired bit $x_i \in \{0,1\}$ from the database while hiding the client's index i from the server, and using total communication that is sublinear in the database size n. A beautiful line of work, starting with that of Chor, Goldreich, Kushilevitz, and Sudan [CGKS95], constructs private information retrieval (PIR) protocols with extremely small communication complexity, either when the client can access multiple non-colluding servers holding replicas of the database [Amb97, CG97,

This is the full version of a paper of the same title at Eurocrypt 2020.

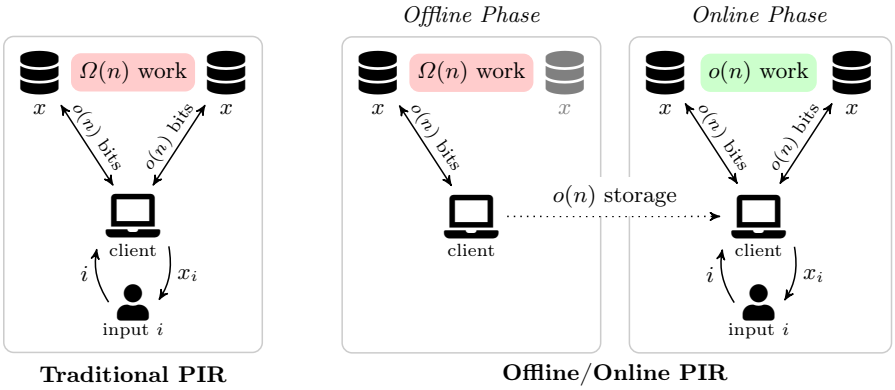

Fig. 1: A comparison of traditional two-server PIR (left) and offline/online PIR with sublinear online time (right). The servers store replicas of a database $x \in \{0,1\}^n$.

BI01, BIKR02, Yek08, Efr12, DG16] or under computational assumptions [KO97, CMS99, KO00, GR05, OS07].

PIR is a fundamental privacy-preserving primitive: it has applications to private messaging [SCM05, AS16, ACLS18], certificate transparency [LG15], private media browsing [GCM+16], online anonymity [MOT+11, KLDF16], privacy-preserving ad targeting [Jue01], and more. In spite of the promise of PIR and the great advances in PIR protocols, there have been essentially no large-scale deployments of PIR technology to date. A primary reason is that while modern PIR protocols have very small *communication* requirements—as small as polylogarithmic in the database size—the *computational* burden they put on the server is still prohibitively expensive.

In particular, in all existing PIR schemes, the work at the servers grows linearly with the database size. That is, the servers essentially take a linear scan over the entire database to respond to each query. Beimel et al. [BIM04] proved that this limitation is in fact inherent: even in the multi-server setting, every secure PIR scheme on an n-bit database must incur $\Omega(n)$ total server-side work. (If the servers probe fewer than n database bits on average in responding to a client's query, then it is likely that the client is reading one of the probed bits.)

This $\Omega(n)$ server-side cost is the major bottleneck for PIR schemes *in theory*, since all other costs in today's PIR protocols (communication, client time, etc.) are sublinear, or even polylogarithmic, in the database size. This $\Omega(n)$ server-side cost is also the major bottleneck for PIR schemes *in practice*, as evidenced by the many heroic efforts to reduce the server-side computational cost in built PIR systems [LG15, AS16, GCM+16, TDG16, ACLS18].

In Section 1.4, we survey the known approaches to reducing the server-side computation in PIR-like schemes. All of these methods increase the storage requirements at the servers and the methods based on standard assumptions (i.e., not requiring obfuscation) increase the required server storage by potentially large polynomial factors. These increased storage costs present new barriers to deployment.

1.1 A new approach: Offline/online PIR with sublinear online time

In this paper, we propose a new approach for reducing the server-side computational burden of PIR. Our idea is to push the (necessary) linear-time server-side computation into a query-independent offline phase, which allows a subsequent online phase to complete in sublinear time. More precisely, we construct PIR schemes in which the client and servers interact in two phases:

- In an **offline** phase, which takes place *before* the client has decided which bit of the database it wants to retrieve, the client fetches a one-time-use "hint" from the database servers.
- In a subsequent **online** phase, which takes place *after* the client has decided which bit of the database it wants to retrieve, the client sends a query to the database servers. Given the servers' answers to this query, along with the hint prefetched earlier, the client can recover its database bit of interest.

Prior work has developed PIR offline/online schemes [DIO01, BIM04, BLW17, PPY18]. In this paper, we construct the first offline/online PIR schemes that simultaneously:

1. run in online time *sublinear* in the database size, and
2. do not increase the storage requirements at the servers.

(See Section 1.4 and Table 2 for a comparison to prior work.) Furthermore, our schemes are based on very simple assumptions—one-way functions in the two-server setting and linearly homomorphic encryption in the single-server setting—and are concretely efficient. The remaining performance bottleneck of our schemes is that one of the servers must perform an amount of *offline* computation in that is *linear* in the database size.

Our schemes advance the state of the art in PIR by enabling two new usage models:

1. **Do the heavy computation in advance.** Our schemes shift the heavy server-side computation out of the critical path of the client's request. For example, we envision deployments of our PIR schemes in which the client and server execute the offline phase overnight, while the user is asleep and when computation is relatively inexpensive. In the morning, when the user wakes up and wants to, say, privately fetch an article from Wikipedia, she can run the online phase to get her article in sublinear time.
   The idea of moving expensive cryptographic work into an input-independent offline phase has seen tremendous success in the setting of multiparty computation [BDOZ11, DPSZ12]. Our schemes achieve the same goal for PIR.
2. **Process a series of queries in sublinear amortized *total* time.** Often, a user wants wants to make a series of adaptive queries to the same database (e.g., as one does when jumping from one Wikipedia article to the next). In this setting, our two-server PIR scheme allows the client to reuse a single hint, fetched in the offline phase, to make *arbitrarily many* adaptive online queries to the database. By reusing the hint, the amortized *total* server-side cost of each query—including both the costs of the offline and online phases—falls to sublinear in the database size. As far as we know, ours is the first PIR scheme

that achieves sublinear amortized total time for adaptive queries without dramatically increasing the client or servers' storage requirements.

1.2 Our results

We give the following results for offline/online private information retrieval with sublinear online time:

Two-server PIR. We give a two server offline/online scheme with sublinear online time. Specifically, for a database consisting of n bits, the offline phase requires the client to interact with one server, which performs $\widetilde{O}(n)$ offline computation. (The notation $\widetilde{O}(\cdot)$ hides arbitrary polylogarithmic factors. In this section, we also elide fixed polynomials in the security parameter.) In the online phase, the client interacts with the second server, which answers the client's query in time $\widetilde{O}(\sqrt{n})$. We give a scheme with statistical security that has total communication $\widetilde{O}(\sqrt{n})$. Assuming that one-way functions exist, the online communication cost falls to $O(\log n)$.

Two-server PIR with sublinear amortized *total* time. We extend our two-server scheme to allow the client to reuse a *single* offline-phase interaction to make a series of polynomially many adaptive online-phase queries. With this scheme, the online cost of each query is still $\widetilde{O}(\sqrt{n})$, but after q online queries, the average *total* computational cost—including the offline-phase computation—falls to $\widetilde{O}(n/q + \sqrt{n})$, or sublinear in the database size.

Single-server PIR. We show how to combine a linearly homomorphic encryption scheme and a standard single-server PIR scheme to obtain a single-server offline/online PIR scheme with sublinear online time. The resulting scheme uses $\widetilde{O}(n^{2/3})$ total communication and the server runs in online time $\widetilde{O}(n^{2/3})$. Furthermore, neither the client nor the server performs any public-key cryptographic operations in the online phase. Under the stronger assumption that fully homomorphic encryption exists, we obtain a single-server scheme with communication and online time $\widetilde{O}(\sqrt{n})$. One drawback is that, unlike its two-server counterpart, our single-server scheme supports only a single online query after each offline interaction, and thus we do not achieve sublinear amortized *total* time. The main benefit of shifting the heavy server-side computation to the offline phase remains.

A lower bound. Finally, we prove a lower bound for offline/online PIR schemes in which the servers store the database in unencoded form and keep no additional state. Specifically, we show that any scheme of this form, that uses C bits of communication in the offline phase, and that probes T bits of the database in the online phase, must satisfy $C \cdot T \geq \widetilde{\Omega}(n)$. This shows that in this model, as far as communication and online server time are concerned, our two-server scheme and the FHE-based single-server scheme are optimal, up to logarithmic factors.

1.3 Limitations

The primary drawback of our new PIR protocols is that they use more total communication than standard PIR schemes do. Today's PIR schemes (with

linear online server-side time) can achieve polylogarithmic communication in the computational setting [CMS99, GR05, IP07, BGI16, DGI+19] and subpolynomial communication ($n^{O(\sqrt{\log\log n/\log n})}$) in the two-server information-theoretic setting [DG16]. In contrast, our schemes with sublinear online time have communication $\widetilde{\Omega}_\lambda(\sqrt{n})$. While we show that it is possible to reduce the *online-phase* communication in the computational setting, our lower bound (Theorem 23) implies that any offline/online PIR scheme with online time $\widetilde{O}(\sqrt{n})$—such as ours—must have $\widetilde{\Omega}(\sqrt{n})$ *total* communication. This limitation is therefore inherent to PIR schemes that have sublinear online server time and in which the servers store the database in unmodified form.

In many settings, we expect that the $\sqrt{n}$ communication cost will be acceptable. Indeed, a number of built systems using PIR [GDL+14, GCM+16, AMBFK16, ACLS18] already suffice with $\sqrt{n}$ communication complexity, since server-side computational cost is the limiting factor. If $\sqrt{n}$ communication is still too high, we show in Corollary 18 that it is possible to amortize the $\sqrt{n}$ offline communication cost of our two-server scheme over polynomially many online reads, each of which requires only *logarithmic* communication. So, our results are still relevant to communication-sensitive settings, when having low amortized complexity is sufficient.

1.4 Related work

Beimel, Ishai, and Malkin [BIM04] proved that the servers in any secure PIR scheme must collectively probe all n bits of the database (on average) to respond to a client's query. We survey the existing strategies for eliminating this key performance bottleneck.

Store the database in encoded form. One ingenious way to circumvent the $\Omega(n)$-server-time lower bound is to have the servers store the database in encoded form. Beimel et al. [BIM04] introduced the notion of *PIR with preprocessing*, in which the servers perform a *one-time* preprocessing of the database $x \in \{0,1\}^n$ and store the database in encoded form $E(x) \in \{0,1\}^N$, where E is a public encoding function and $N \gg n$. In the two-server setting, their PIR schemes with preprocessing achieve $n^{1/2+\epsilon}$ total communication and $n^{1/2+\epsilon}$ server-side time, for any $\epsilon > 0$. The downside of this approach is that the server-side encoding can be quite large. For example, to achieve $n^{0.6}$ server-side time and communication using their two-server scheme requires the server to store an encoded database of size $N = n^{3.2}$. Even for modest database sizes (e.g., $n \approx 2^{20}$), the encoded database would be much too large to materialize in practice (many petabytes). While it would be fascinating to construct improved schemes for two-server PIR with preprocessing—perhaps with encoding size $N = 10n$ and online time and communication $n^{1/3}$—this goal appears far out of reach.

The schemes of Beimel et al. apply only to the multi-server setting. Two recent works [BIPW17, CHR17] study *doubly efficient PIR*, which are in some sense single-server PIR-with-preprocessing schemes. In the *designated-client* model of doubly efficient PIR, the client encodes the database using a long-term secret

Table 2: A comparison of PIR schemes when cast into the offline/online model, on database size n, in which each client makes q adaptive online queries, and in which m clients execute the offline phase before the first client executes the online phase.

- The offline and online costs are *per-query* costs. Thus, if a scheme has a offline phase of server cost n, which can be reused over q online queries, we write its per-query offline cost as n/q. If a scheme has a *one-time* offline phase that can be reused for an unbounded number of clients and queries (as in [BIM04, BIPW17]), we view the scheme as having zero offline cost.
- The extra storage cost is the number of bits, in addition to the database, that client and server must hold between the offline and online phases.

All columns omit $\mathrm{poly}(\lambda)$ factors, for security parameter λ, and also low-order $\mathrm{polylog}(n)$ factors. Here, $\epsilon > 0$ is an arbitrarily small constant and c refers to some constant in $\mathbb{N}$.

		Offline			**Online**			**Extra storage**	
		Time		Comm.	Time		Comm.		
	Assumption	Client	Server		Client	Server		Client	Server
Two-server									
[DG16]	None	0	0	0	$n^{o(1)}$	n	$n^{o(1)}$	0	0
[BGI16]	OWF	0	0	0	$\log n$	n	$\log n$	0	0
[BLW17]*	LWE	$\log n$	n	$\log n$	$\log n$	n	$\log n$	$\log n$	0
[BIM04]	None	0	0	0	$n^{0.9}$ $n^{0.6}$ $n^{0.55}$	$n^{0.9}$ $n^{0.6}$ $n^{0.55}$	$n^{0.9}$ $n^{0.6}$ $n^{0.55}$	0	$n^{1.27}$ $n^{3.2}$ $n^{37.7}$
[PR93]	None	n	n	n	$\log n$	n	n	0	0
[DIO01]†	OWF	0	n	0	$\log n$	$\log n$	$\log n$	0	mn
Thm. 11	None	$n^{1/2}$	n	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	0
Thm. 14	OWF	$n^{1/2}$	n	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	$\log n$	$n^{1/2}$	0
Thm. 17	OWF	$\frac{n^{1/2}}{q}$	n/q	$\frac{n^{1/2}}{q}$	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	0
Single-server									
[KO97]	Lin. hom. enc.	0	0	0	n^{ϵ}	n	n^{ϵ}	0	0
[CMS99]	ϕ-hiding	0	0	0	$\log^{c} n$	n	$\log^{c} n$	0	0
[Lip05]	DCR	0	0	0	$\log^{c} n$	n	$\log^{2} n$	0	0
[Lip09]‡	Lin. hom. enc.	0	n	0	$\log n$	n	$\log n$	0	n
[PPY18]‡	Any PIR	n/q	n/q	n/q	n	n	$\log^{c} n$	$n^{1/2}$	0
[BIPW17]	OLDC	n/q	n/q	n/q	n^{ϵ}	n^{ϵ}	n^{ϵ}	c	mn
[CHR17]	OLDC	n/q	n/q	n/q	n^{ϵ}	n^{ϵ}	n^{ϵ}	c	mn
[BIPW17]§	OLDC+**VBB Obf.**	0	0	0	n^{ϵ}	n^{ϵ}	n^{ϵ}	0	n
Thm. 20	Lin. hom. enc.	$n^{2/3}$	n	$n^{2/3}$	$n^{2/3}$	$n^{2/3}$	$n^{1/3}$	$n^{2/3}$	0
Thm. 22	FHE	$n^{1/2}$	n	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	$n^{1/2}$	0
Lower bound (For any $1 \le \beta \le n$.)									
[BIM04]		–	n/β	–	–	–	–	–	β
Thm. 23		–	–	n/β	–	β	–	–	0¶

* Based on constrained PRFs, which initially required multilinear maps, but later constructed from standard assumptions [BKM17, CC17, BTVW17].

† A scheme combining ideas from [DIO01, BIM04, BGI16] can acheive these parameters [Ish19].

‡ Requires only a sublinear number of public-key operations.

§ Requires that a trustworthy party encodes the database using secret randomness.

¶ Our lower bound holds only for PIR schemes that store the database in its native form.

key (hidden from the server) and stores the encoded database on the server. Under a new cryptographic assumption, the client can subsequently privately query this encoded database many times, and the server can answer the query

in time sublinear in the database size. In the *public-key* analogue of doubly efficient PIR, a server that stores a single database encoding enables multiple mutually distrusting clients to query the database using a short public key. Boyle et al. [BIPW17] construct a public-key doubly efficient PIR scheme with sublinear query time, under a new cryptographic assumption and in a model with virtual black-box obfuscation.

Hamlin et al. [HOWW18] introduce a notion of *private anonymous data access* ("PANDA") schemes, in which many clients can access an encoded database such that (1) as in standard PIR schemes, the server does not learn which bits of the database a client is reading and (2) the server can respond to a client's request in time sublinear—even polylogarithmic—in the database size. Unlike in doubly efficient PIR schemes, the server in PANDA may store mutable state. Hamlin et al. give an instantiation of a PANDA scheme from fully homomorphic encryption [Gen09]. A limitation of the existing PANDA schemes is that they require the server storage and time to grow with the number of malicious clients interacting with the system. In our setting, in which the number of malicious clients could be unbounded, the storage and online server time of a PANDA scheme would also be unbounded.

The general framework of PIR with preprocessing is extremely promising, since preprocessing schemes can plausibly allow both *polylogarithmic* total communication and total work—which is impossible in the offline/online setting. That said, these preprocessing schemes necessarily increase the storage costs at the servers, by large polynomial factors in many cases. The single-server preprocessing schemes additionally rely on relatively heavy cryptographic assumptions. In contrast, in our schemes, the servers store the database x in *unencoded* form and keep no additional state. The trade-off is that, in our schemes, the client and servers must run the linear-server-time offline phase *once per client* (Section 4) or *once per query* (Sections 3 and 5).

Use *linear* additional storage *per query*. Beimel, Ishai, and Malkin [BIM04, Section 7.2], building on earlier work of Di Crescenzo, Ishai, and Ostrovsky [DIO98, DIO01], give an alternative way to reduce the server-side online time in PIR. In their model, the client submits a request to the servers in an offline phase. The servers use this request to generate a one-time-use n-bit encoding of the n-bit database, which the servers store. In a subsequent online phase, the client can privately query the servers for a database bit and the servers use their precomputed encoding to respond in sublinear online time. The total communication and online server-side work in these schemes can be as low as $\mathrm{polylog}(n)$ [Ish19]. However, the server-side storage costs can be large: for *each client*, the servers must store n additional bits until that client makes its online query. If m clients concurrently access the database, the storage requirements at the servers increase to mn bits. (In contrast, the schemes in our work require no extra server-side storage.)

Use *linear* online time. Another line of work reduces the server-side computational burden of PIR protocols by working in the offline/online model we consider. To our knowledge, all prior protocols in the offline/online model require *linear online time* at the servers.

Boneh, Lewi, and Wu [BLW17] show that "privately constrained PRFs" imply a two-server online/offline scheme in which only one of the servers needs to be active in the online phase. The scheme has polylogarithmic communication complexity, yet the online server's work is *linear* in the database size. Subsequent work [BKM17, CC17, BTVW17] constructs such PRFs from standard lattice assumptions.

Towards reducing the server's computation time in PIR protocols, Patel, Persiano, and Yeo [PPY18] introduce the notion of *private stateful information retrieval*. They give single-server schemes in which, after an offline phase, the client can privately retrieve a bit from the database while requiring the server to only perform a number of online public-key operations sublinear in the database size, along with a linear number of symmetric-key operations. The offline phase of their protocol requires the client to download a linear number of bits in the offline phase and the server must perform a linear number of total operations in the online phase. Their schemes do allow amortizing the linear-communication offline phase over multiple subsequent queries, although the online time is always linear. In contrast, our protocols have total communication and online time *sublinear* in the database size, even for a single query.

Demmler, Herzberg, and Schneider [DHS14] give a scheme which reduces the computational burden of each server by means of sharding the database. The combined work of all servers in their scheme is still linear.

Marginally sublinear online time. The original PIR paper [CGKS95] points out that a three-party communication protocol of Pudlák and Rödl [PR93, Theorem 3.5] yields a two-server PIR protocol. (See also the subsequent journal version [PRS97].) In particular, on an n-bit database, that protocol has total communication $\alpha(n) = O(n\frac{\log\log n}{\log n})$, or just slightly sublinear. Closer inspection of this protocol reveals that one of the two servers can additionally be made to run in *sublinear time* $\alpha(n)$, and thus this early scheme can be cast as an offline/online PIR scheme with just slightly sublinear offline communication. As far as we know, no prior work has drawn attention to this fact.

Lipmaa [Lip09] constructs a computational single-server PIR protocol with preprocessing. In its offline phase, the server encodes the database as a branching program with $(n + o(n))/\log n$ nodes, and stores the branching program, using $n + n/\operatorname{polylog}(n)$ bits. In the online phase, the server homomorphically evaluates the branching program, using a protocol of Ishai and Paskin [IP07], which requires $O(n/\log n)$ public key operations, or slightly sublinear in the database size. (The homomorphic ciphertexts must be no shorter than the security parameter $\lambda = \omega(\log n)$, and so, strictly speaking, the number of bit operations in the online phase is still linear. However, the running time is dominated by the public key operations.)

The complexity of these two protocols is much larger than ours but we still find it interesting to see such radically different ways to construct two-server PIR with sublinear online time.

Amortize work. It is also possible to improve the computational efficiency of PIR by having each PIR server jointly process a batch of queries. If a server

can process a batch of Q queries to an n-bit database at $o(Qn)$ cost, processing queries in a batch yields sublinear amortized time per query at the server. This general strategy is fruitful both when the batched queries originate from the same client [IKOS04, Hen16, ACLS18] and from different clients [BIM04, IKOS06, LG15].

Our multi-query scheme of Section 4 similarly allows the client to amortize the server's linear-time offline computation over many queries—as in batch PIR. The difference is that our multi-query scheme allows the client to make its queries *adaptively* (one at a time), while batch-PIR schemes require the client to make all queries in a batch *non-adaptively* (all at once).

Relax the security property. One final approach to reducing the online server time in PIR is to aim for a weaker security property than standard cryptographic PIR schemes do. Toledo, Danezis, and Goldberg give PIR schemes with a differential-privacy-style notion of security and show that when some leakage of the client's query to the server is allowed, the servers can run in sublinear online time [TDG16].

1.5 Technical overview

To illustrate our techniques, we start by presenting a simplified version of our two-server offline/online PIR scheme with statistical security. The online phase of this protocol runs in time $o(n)$, and the protocol's total communication is $o(n)$.

A toy protocol. Two servers hold a replica of the database $x \in \{0,1\}^n$. The two phases of the protocol proceed as follows:

Offline phase. This phase takes place before the client has decided which bit it wants to read from the database.

- The client divides the database indices $\{1, \ldots, n\}$ at random into $\sqrt{n}$ disjoint sets $(S_1, \ldots, S_{\sqrt{n}})$, each of size $\sqrt{n}$, and sends these sets to the first server. (Sending these sets explicitly would take $\Omega(n \log n)$ communication, which is too much. We explain later how to reduce the communication in this step.)
- The first server receives the sets $(S_1, \ldots, S_{\sqrt{n}})$ from the client. For each such set S_j, it computes the parity of the database bits indexed by the set. That is, for $j \in \{1, \ldots, \sqrt{n}\}$, the server computes the parity $h_j \leftarrow \sum_{i \in S_j} x_i \bmod 2$. The server sends these parity bits $(h_1, \ldots, h_{\sqrt{n}})$ to the client.
- The client stores the sets $(S_1, \ldots, S_{\sqrt{n}})$ and the parity bits $(h_1, \ldots, h_{\sqrt{n}})$.

Online phase. This phase begins once the client has decided on the index $i \in [n]$ of the bit it wants to read from the database.

- The client finds the set S_j that contains its desired index i. The client then removes a single item i^* from the set S_j, which it chooses as follows:
  - With probability $1 - (\sqrt{n} - 1)/n$ the client chooses $i^* \leftarrow i$.
  - With the remaining probability, the client chooses i^* randomly from the set of all other elements in S_j.

  The client then sends the set $S' \leftarrow S_j \setminus \{i^*\}$ to the second server.

– Upon receiving the set S' from the client, the second server computes and sends back to the client the parity of the database bits indexed by the set: $a \leftarrow \sum_{i \in S'} x_i \bmod 2$. Computing the answer requires the second server to probe at most $|S'| = O(\sqrt{n})$ bits of the database, which allows the server to run in only $\widetilde{O}(\sqrt{n})$ time.
– Finally, when the client receives the answer from the second server, it recovers the value of the database bit x_{i^*} by computing $x_{i^*} \leftarrow h_j - a \bmod 2$. Crucially, since the client has chosen i^* with a bias towards i, it recovers the value x_i of its bit of interest with high probability $1 - O(1/\sqrt{n})$. (By iterating the scheme λ times in parallel, the client can drive the failure probability down to at most $2^{-\lambda}$.)

With a bit of work, it is possible to show that the set S' that the client sends to the second server is a uniformly random subset of $[n]$ of size $\sqrt{n} - 1$. Thus, the values that both servers see are distributed independently of the index i that the client is trying to read.

The resulting scheme already achieves the main goal of interest: in the online phase, the server can respond to the client's query in time $O(\sqrt{n})$. However, the toy scheme also has two major shortcomings:

1. The communication in the *offline* phase is *super-linear*: sending the sets $(S_1, \ldots, S_{\sqrt{n}})$ to the first server requires $\Omega(n \log n)$ bits.
2. The scheme requires $\Theta(n \log n)$ bits of client storage between the offline phase and the online phase.

We can address both of these challenges at once by partially derandomizing the client. In the revised scheme, in the offline phase, the client chooses a *single* set $S \subseteq [n]$ of size $\sqrt{n}$. The client also sends to the server $\sqrt{n}$ random "shifts" $\Delta = \{\delta_1, \delta_2, \ldots, \delta_{\sqrt{n}}\} \in [n]$. The client and server then use S and Δ to construct a collection of $\sqrt{n}$ sets $(S_1, \ldots, S_{\sqrt{n}})$ by setting, for every $j \in \{1, \ldots, \sqrt{n}\}$, $S_j \leftarrow \{i + \delta_j \mid i \in S\}$. The client and the server then run the rest of the toy protocol using this collection of sets. This modification increases the failure probability, since there is now some chance that the client's desired index i will not be in any of the sets $(S_1, \ldots, S_{\sqrt{n}})$. Even so, the client and servers can repeat the protocol $O(\log n)$ times in parallel to drive down the failure probability.

This modifications reduces both the communication complexity of the offline phase and the amount of client storage and time to $\widetilde{O}(\sqrt{n})$. With some work, we can also argue that this modification preserves security.

Improvements to the toy scheme. While the above patched two-server scheme achieves all of our efficiency goals, it leaves a few things to be desired:

– *Reducing online communication with puncturable pseudorandom sets.* In the protocol sketched above, the communication in the online phase is $\Theta(\sqrt{n})$. Under the assumptions that one-way functions exist, we can reduce the online-phase communication to $\mathrm{poly}(\lambda, \log n)$, for security parameter λ.

  To do so, we introduce a new tool, which we call a *puncturable pseudorandom set* (Section 2). Essentially, a puncturable pseudorandom set allows the client in the toy scheme above to send the server a compressed representation of

the random set S, in the form of a short key k. Furthermore, the set key is "puncturable," in that for any $i^* \in S$, the client can produce a punctured set key k_{i^*} that is a compressed representation of $S \setminus \{i^*\}$. Crucially, the punctured key k_{i^*} also hides the identity of the removed element i^*.

We construct a puncturable pseudorandom set from puncturable PRFs [BW13, KPTZ13, BGI14, SW14] (Theorem 3), which have simple constructions from pseudorandom generators. The keys in our construction have size $O(\lambda \log n)$ for sets of size $O(\sqrt{n})$ over a universe of size n and security parameter λ. Plugging this puncturable pseudorandom set construction into the toy scheme above reduces the communication complexity of the online phase to the length of a single punctured set key, plus the single bit answer, for $O(\lambda \log n)$ bits total.

– *Refreshing the client's state.* The client in the toy scheme can only use the results of the (computationally expensive) offline phase to read a *single* bit from the database. The following modification to the toy scheme allows the client to "refresh" the bits it downloads in the offline phase, so that it can reuse these bits for many online queries (Section 4).

  After the client makes a query for index $i \in [n]$ using set S_j, the client discards that set from its state. Now the client must somehow "refresh" its local state. Our observation is that the set S_j is a random size-$\sqrt{n}$ subset of $[n]$, conditioned on $i \in S_j$. The client refreshes its state by asking the first server for the parity of a random size-$(\sqrt{n}-1)$ subset S' of $[n]$. Since the client already knows the value of x_i, it can compute and store the parity of the database bits in the set $S' \cup \{i\}$. (Ensuring that this refreshing process maintains security requires handling some technicalities.)

  Although this construction requires the client to use *independent* random sets $(S_1, \ldots, S_{\sqrt{n}})$, using puncturable pseudorandom sets the client can send to the offline server all of them using only $\widetilde{O}(\sqrt{n})$ bits of communication.

– *From two servers to one.* Converting the two-server offline/online PIR scheme to a single-server one is conceptually simple. Say that in the offline phase of the two-server scheme, the client sends a query q to the first server and receives an answer a. To convert it into a single-server scheme, we have the client send an encryption $E(q)$ of its offline query to the server, and we have the server homomorphically compute and send back the encrypted answer $E(a)$. Since the server learns nothing about the offline query q, the online phase can proceed exactly as in the two-server scheme.

  With fully homomorphic encryption [Gen09], this transformation is straightforward and maintains the communication complexity of the original two-server scheme. We show in Theorem 20 that it is possible to execute these steps using the much lighter-weight tools of linearly homomorphic encryption and single-server PIR, with slightly worse communication efficiency and online time: $\widetilde{O}(n^{2/3}) \cdot \mathrm{poly}(\lambda)$, for security parameter λ.

– *Proving optimality.* Finally, we prove a lower bound on the offline communication and online time using a classic lower bound of Yao [Yao90]. In particular,

we show (Lemma 50) that any offline/online PIR scheme with small offline communication and online time, and in which the servers store the database in unmodified form, implies a good solution to "Yao's Box Problem." We then apply a preexisting time/space lower bound against algorithms for Yao's Box Problem to complete the lower bound (Theorem 23).

1.6 Notation

We use $\mathbb{N}$ to denote the set of positive integers. For an integer $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$ and 1^n denotes the all-ones binary string of length n. For $n \in \mathbb{N}$ and $s \in [n]$, an s-subset of $[n]$ is a subset of size exactly s, and $\binom{[n]}{s}$ denotes the set of all s-subsets of $[n]$. Logarithms are taken to the base 2. We ignore integrality concerns and treat expressions like $\sqrt{n}$, $\log n$, and m/n as integers.

The expression $\text{poly}(\cdot)$ refers to a fixed (unspecified) polynomial in its parameter. The notation $\widetilde{O}(\cdot)$ hides arbitrary polylogarithmic factors, i.e., $f(n) = \widetilde{O}(g(n))$ if $f(n) = O(g(n)) \cdot \text{poly}(\log n)$. The notation $O_\lambda(\cdot)$ hides arbitrary polynomial factors in (the security parameter) λ, i.e., $f(n, \lambda) = O_\lambda(g(n))$ if $f(n, \lambda) = O(g(n)) \cdot \text{poly}(\lambda)$.

For a finite set S, the notation $x \xleftarrow{R} S$ refers to choosing x independently and uniformly at random from the set S. For a distribution $\mathcal{D}$ over a set S, the notation $x \xleftarrow{R} \mathcal{D}$ refers to choosing $x \in S$ according to distribution $\mathcal{D}$. For $p \in [0, 1]$, the notation $b \xleftarrow{R} \mathsf{Bernoulli}(p)$ refers to choosing the bit b to be '1' with probability p and '0' with probability $1 - p$.

We use the RAM model of computation with the size of the word logarithmic in the input length and linear in the security parameter. To avoid dependence on the specifics of the computational model, we usually specify running times up to polylogarithmic factors. Throughout this text, an efficient algorithm is a probabilistic polynomial time algorithm. Furthermore, we allow all adversaries to be non-uniform. (Though this is not fundamental, and, with appropriate modifications in the security games, the results hold also in the uniform setting.)

We say that a pseudorandom generator (PRG) or pseudorandom permutation (PRP) is ϵ-secure if no efficient adversary can distinguish the PRG or PRP from random with advantage better than $\epsilon(\lambda)$, on security parameter λ.

2 Puncturable pseudorandom sets

In this section, we introduce a new cryptographic primitive called *puncturable pseudorandom sets* and give few natural constructions. Puncturable pseudorandom sets are a key component of our PIR schemes.

A puncturable pseudorandom set is very closely related to a puncturable pseudorandom function ("puncturable PRF") [BW13, KPTZ13, BGI14, SW14, HKW15]. To explain the difference by analogy: a PRF key is a compressed representation of a function $f : [n] \to [n]$, and a PRF key punctured at point $x^* \in [n]$ allows its holder to evaluate f at every point in $[n]$ *except* at the

punctured point x^*. The punctured key should reveal nothing about the value of $f(x^*)$ to its holder. (The formal standard definition appears in Appendix A.2.)

Analogously, the key for a *puncturable pseudorandom set* is a compressed representation of a a pseudorandom set $S \subseteq [n]$. The set key punctured at element $x^* \in S$ allows its holder to recover all elements of S *except* the punctured element x^*. The punctured set key reveals nothing about x^* to its holder, apart from that fact that x^* is not one of the remaining elements in S.

2.1 Definitions

Let $s : \mathbb{N} \to \mathbb{N}$ be a function such that $s(n) \leq n$. A *puncturable pseudorandom set* with set size s consists of a key space $\mathcal{K}$, a punctured-key space $\mathcal{K}_\mathsf{p}$, and a triple of algorithms:

- $\mathsf{Gen}(1^\lambda, n) \to \mathsf{sk}$, a randomized algorithm that takes as input the security parameter $\lambda \in \mathbb{N}$, expressed in unary, and a universe size $n \in \mathbb{N}$, expressed in binary, and outputs a set key $\mathsf{sk} \in \mathcal{K}$,
- $\mathsf{Punc}(\mathsf{sk}, i) \to \mathsf{sk}_\mathsf{p}$, a deterministic algorithm that takes in a key $\mathsf{sk} \in \mathcal{K}$ and an element $i \in [n]$, and outputs a punctured set key $\mathsf{sk}_\mathsf{p} \in \mathcal{K}_\mathsf{p}$, and
- $\mathsf{Eval}(\mathsf{sk}) \to S$, a deterministic algorithm that takes in a key $\mathsf{sk} \in \mathcal{K} \cup \mathcal{K}_\mathsf{p}$ and outputs a description of a set $S \subseteq [n]$, written as $|S|$ strings of $\log n$ bits in length each.

A puncturable pseudorandom set must satisfy the following notions of *efficiency*, *correctness* and *security*.

Efficiency. For every security parameter $\lambda \in \mathbb{N}$ and universe size $n \in \mathbb{N}$, the routines Gen, Punc, and Eval run in time $s(n) \cdot \mathrm{poly}(\lambda, \log n)$, where $s(n)$ is the set size.

Correctness. For every $\lambda, n \in \mathbb{N}$, if one samples $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)$ and computes $S \leftarrow \mathsf{Eval}(\mathsf{sk})$, it holds, with probability 1 over the randomness of Gen, that

1. $S \in \binom{[n]}{s(n)}$, where $\binom{[n]}{s(n)}$ denotes the set of all size-$s(n)$ subsets of $[n]$, and
2. for all $i \in S$, $\mathsf{Eval}(\mathsf{Punc}(\mathsf{sk}, i)) = S \setminus \{i\}$.

Security. Let Ψ be a puncturable pseudorandom set with set size $s \colon \mathbb{N} \to \mathbb{N}$. Let $W_{\lambda,n}$ be the event that adversary $\mathcal{A}$ wins in Game 1 with respect to Ψ, with security parameter λ and universe size n. Then we define $\mathcal{A}$'s guessing advantage as:

$$\mathsf{PSAdv}[\mathcal{A}, \Psi](\lambda, n) := \Pr[W_{\lambda,n}] - \frac{1}{n - s(n) + 1}. \tag{1}$$

A puncturable pseudorandom set Ψ is *computationally secure* if for every $\lambda \in \mathbb{N}$, every polynomially bounded $n = n(\lambda)$, and every non-uniform polynomial-time adversary $\mathcal{A}$, we have that $\mathsf{PSAdv}[\mathcal{A}, \Psi](\lambda, n) \leq \mathrm{negl}(\lambda)$. The puncturable pseudorandom set is ϵ-secure if that advantage is smaller than $\epsilon(\lambda, n)$. We say that Ψ is *perfectly secure* if for every $\lambda, n \in \mathbb{N}$ and for every (computationally unbounded) adversary $\mathcal{A}$, we have that $\mathsf{PSAdv}[\mathcal{A}, \Psi](\lambda, n) = 0$.

Game 1 (Puncturable pseudorandom set security). For $\lambda, n \in \mathbb{N}$, and a puncturable pseudorandom set $\Psi = (\mathsf{Gen}, \mathsf{Punc}, \mathsf{Eval})$, we define the following game, played between a challenger and an adversary:

- The challenger executes the following steps:
  - $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)$
  - $S \leftarrow \mathsf{Eval}(\mathsf{sk})$
  - $x^* \xleftarrow{R} S$
  - $\mathsf{sk}_\mathsf{p} \leftarrow \mathsf{Punc}(\mathsf{sk}, x^*)$

  and sends 1^λ and sk_p to the adversary.
- The adversary outputs an integer $x' \in [n]$.

We say that the adversary "wins" if $x^* = x'$.

In Appendix B.1, we show that this security property implies that the output of Eval on a random key is a pseudorandom set in $\binom{[n]}{s(n)}$.

Throughout this work, we often refer to puncturable pseudorandom sets as *puncturable pseudorandom sets* for brevity.

2.2 Constructions

Fact 2 (Perfectly secure puncturable pseudorandom set with linear-sized keys). *For any function $s : \mathbb{N} \to \mathbb{N}$ with $s(n) \leq n$, there is a perfectly secure puncturable pseudorandom set with set size s. Moreover, for universe size n, the set keys and punctured keys are both of length $(s(n) + O(1)) \log n$ bits.*

Proof. The set key is the description of a set $S \xleftarrow{R} \binom{[n]}{s}$—written as s numbers, each of $\log n$ bits in length, along with a description of the universe size n. A punctured key is just this set of elements with the punctured element removed. □

Theorem 3 (puncturable pseudorandom set with short keys from puncturable PRFs). *Suppose there exists an $\epsilon_\mathcal{F}$-secure puncturable PRF (as in Appendix A.2) that, on security parameter λ and input-space size n, has keys of length $\kappa(\lambda, n)$ bits and punctured keys of length $\kappa_\mathsf{p}(\lambda, n)$ bits. Then, there exists an ϵ-secure puncturable pseudorandom set with set size $\Theta(\sqrt{n})$ that, on security parameter λ and universe size n, has*

- *set keys of length $\kappa(\lambda, n) + O(\log n)$ bits and*
- *punctured keys of length $\kappa_\mathsf{p}(\lambda, n) + O(\log n)$ bits, and*
- *$\epsilon(\lambda, n) = \mathrm{poly}(\lambda, n) \cdot (\epsilon_\mathcal{F} + 2^{-\lambda})$.*

A puncturable pseudorandom set that proves the theorem appears in Construction 4. We prove security and correctness of the construction in Appendix B.2.

Remark 5. The Gen routine in Construction 4 fails with negligible probability, and therefore, as presented, the construction has imperfect correctness. We can achieve perfect correctness by having the Eval and Punc routines treat $\mathsf{sk} = \bot$ as some fixed set (e.g., the set $[s]$). Our security analysis accounts for this.

Instantiating Theorem 3 with the puncturable PRF [BW13, KPTZ13, BGI14] based on the tree-based PRF of Goldreich, Goldwasser, and Micali [GGM86] leads

Construction 4 (Puncturable pseudorandom set from puncturable PRF). Given a puncturable PRF $\mathcal{F} = (\mathsf{PRFGen}, \mathsf{PRFPunc}, \mathsf{PRFEval})$, we construct a puncturable pseudorandom set $\Psi_{\mathcal{F}} = (\mathsf{Gen}, \mathsf{Punc}, \mathsf{Eval})$ with set size $s(n) := \sqrt{n}/2$.

$\underline{\Psi_{\mathcal{F}}.\mathsf{Gen}(1^\lambda, n) \to \mathsf{sk}}$

- Repeat at most λ times:
  - Sample $\mathsf{k} \leftarrow \mathsf{PRFGen}(1^\lambda, n)$.
  - Compute $S \leftarrow \{\mathsf{PRFEval}(\mathsf{k}, 1),\ \mathsf{PRFEval}(\mathsf{k}, 2),\ \ldots\ ,\ \mathsf{PRFEval}(\mathsf{k}, s(n))\}$.
  - If $|S| = s(n)$, halt and output $\mathsf{sk} \leftarrow (n, \mathsf{k})$. output $\perp$.
- After running λ iterations of the loop unsuccessfully, output $\perp$.

$\underline{\Psi_{\mathcal{F}}.\mathsf{Punc}(\mathsf{sk}, i) \to \mathsf{sk}_\mathsf{p}}$

- Parse the secret key as a pair (n, k).
- Find the least integer ℓ such that $\mathsf{PRFEval}(\mathsf{k}, \ell) = i$.
  If no such ℓ exists, output $\perp$.
- Compute $\mathsf{k}_\mathsf{p} \leftarrow \mathsf{PRFPunc}(\mathsf{k}, \ell)$ and output $\mathsf{sk}_\mathsf{p} \leftarrow (n, \mathsf{k}_\mathsf{p})$.

$\underline{\Psi_{\mathcal{F}}.\mathsf{Eval}(\mathsf{sk}) \to S}$

- Parse the secret key as a pair (n, k).
- Output the set $S \leftarrow \{\mathsf{PRFEval}(\mathsf{k}, 1),\ \mathsf{PRFEval}(\mathsf{k}, 2),\ \ldots\ ,\ \mathsf{PRFEval}(\mathsf{k}, s(n))\}$.
- (If k is punctured at some value, skip this value when computing S.)

to a very efficient puncturable pseudorandom set construction from pseudorandom generators. In Appendix B.3, we prove the following:

Corollary 6. *Assuming that pseudorandom generators (PRGs) exist, there exists a secure puncturable pseudorandom set with set size $\Theta(\sqrt{n})$.*

In particular, for every ϵ_G-secure length-doubling PRG G, there exists an ϵ-secure puncturable pseudorandom set Ψ_G with set size $\sqrt{n/2}$, that has, for every security parameter $\lambda \in \mathbb{N}$ and universe size n,

- *set keys of $\lambda + O(\log n)$ bits in length,*
- *punctured keys of $O(\lambda \log n)$ bits in length, and*
- *$\epsilon(\lambda, n) \leq \mathrm{poly}(\lambda, n) \cdot (\epsilon_G(\lambda) + 2^{-\lambda})$.*

A puncturable pseudorandom set with fast membership testing from PRPs. We say that a puncturable pseudorandom set Ψ on universe size n has a *fast membership test* if there exists an algorithm InSet that takes as input a set key sk and an element $i \in [n]$, runs in time $\mathrm{poly}(\lambda, \log n)$, and outputs "1" if $i \in \Psi.\mathsf{Eval}(\mathsf{sk})$ and "0" otherwise. Crucially, the running time of the fast membership test must grow only with $\log n$, rather than linearly with the set size $s(n)$. The following is a construction of such a puncturable pseudorandom set. The proof appears in Appendix B.4.

Theorem 7. *Suppose there exists an ϵ_P-secure pseudorandom permutation that, on security parameter λ and input-space size n, has keys of length $\kappa(\lambda, n)$ bits.*

Then, there exists an ϵ-secure puncturable pseudorandom set for any set size $s : \mathbb{N} \to \mathbb{N}$ that, on security parameter λ and universe size n, has

- *set keys of length $\kappa(\lambda, n)$ bits,*
- *punctured keys of length $s \cdot O(\log n)$ bits,*
- *$\epsilon \leq \text{poly}(\lambda, n) \cdot \epsilon_P$, and*
- *a fast membership test.*

2.3 Shifting puncturable pseudorandom sets

When using puncturable pseudorandom sets in this paper, we will want to equip them with two additional functionalities.

1. $\mathsf{GenWith}(1^\lambda, n, i) \to \mathsf{sk}$ is an algorithm that takes in $n \in \mathbb{N}$ and $i \in [n]$, and outputs a uniformly random puncturable pseudorandom set key sk for a $s(n)$-subset of $[n]$, subject to the constraint that $i \in \mathsf{Eval}(\mathsf{sk})$.
2. $\mathsf{Shift}(\mathsf{sk}, \delta) \to \mathsf{sk}'$ is an algorithm that takes in a set key $\mathsf{sk} \in \mathcal{K}$ and an integer $\delta \in [n]$, and outputs a set key sk' such that $\mathsf{Eval}(\mathsf{sk}') = \{i + \delta \mid i \in \mathsf{Eval}(\mathsf{sk})\}$. (The addition $i + \delta$ is done modulo n, and we identify $0 \in \mathbb{Z}_n$ with $n \in [n]$.)

In Appendix B.5, we show how to extend any puncturable pseudorandom set to efficiently support both these functionalities by including a shift $\Delta \in [n]$ with every key and interpreting every element i in the base set as $(i + \Delta) \bmod n$ in the encompassing set. This transformation only increases the size of the puncturable set keys by an additive $O(\log n)$ term. Therefore, we subsequently assume without a loss of generality that every puncturable set is equipped with $\mathsf{GenWith}$ and Shift.

3 Two-server PIR with sublinear online time

We now formally define two-server offline/online PIR and construct such schemes that achieve sublinear online time and provide either statistical or computational security.

3.1 Definition

Informally, a two-server offline/online PIR scheme is a protocol between a client, an offline server, and an online server. Both servers have access to a database $x \in \{0,1\}^n$. The PIR protocol proceeds in five steps:

1. First, the client uses the Setup algorithm to generate its own *client key* ck, along with a hint request q_h. The client sends the hint request q_h to the offline server. Crucially, the client can run the Setup algorithm *before* it has decided which bit of the database it wants to read.
2. The offline server feeds the hint request q_h and the database $x \in \{0,1\}^n$ into the Hint algorithm, which generates a hint h that the offline server returns to the client.
3. Once the client has decided on the index $i \in [n]$ of the bit it wants to read from the database, it feeds its key ck and index i into the Query algorithm, which produces a query q. The client sends this query to the online server.

4. The online server feeds the client's query q into the Answer algorithm that is further given access to the database. (The focus is on schemes in which the Answer algorithm probes $o(n)$ bits of the database and run in time $o(n)$.) The online server then returns the answer a to the client.
5. The client feeds the hint h and the answer a into algorithm $\mathsf{Reconstruct}$, which outputs the i-th bit of the database.

A *secure* offline/online PIR scheme should guarantee that neither server independently learns anything (in either a statistical or computational sense) about the client's private index i.

Definition 8 (Offline/online PIR). An offline/online PIR scheme is a tuple $\Pi = (\mathsf{Setup}, \mathsf{Hint}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ of five efficient algorithms:

- $\mathsf{Setup}(1^\lambda, n) \to (\mathsf{ck}, q_h)$, a randomized algorithm that takes in security parameter λ and database length n and outputs a client key ck and a hint request q_h.
- $\mathsf{Hint}(x, q_h) \to h$, a deterministic algorithm that takes in a database $x \in \{0,1\}^n$ and a hint request q_h and outputs a hint h,
- $\mathsf{Query}(\mathsf{ck}, i) \to q$, a randomized algorithm that takes in the client's key ck and an index $i \in [n]$, and outputs a a query q,
- $\mathsf{Answer}^x(q) \to a$, a deterministic algorithm that takes as input a query q and gets access to an oracle that:
  - takes as input an index $j \in [n]$, and
  - returns the j-th bit of the database $x_j \in \{0,1\}$,

  outputs an answer string a, and
- $\mathsf{Reconstruct}(h, a) \to x_i$, a deterministic algorithm that takes as a hint h and an answer a, and outputs a bit x_i.

Furthermore, the scheme Π must satisfy the following properties:

Correctness. For every $\lambda, n \in \mathbb{N}$, $x \in \{0,1\}^n$, and $i \in [n]$, we require that

$$\Pr\left[\mathsf{Reconstruct}(h,a) = x_i : \begin{array}{r}(\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n) \\ h \leftarrow \mathsf{Hint}(x, q_h) \\ q \leftarrow \mathsf{Query}(\mathsf{ck}, i) \\ a \leftarrow \mathsf{Answer}^x(q)\end{array}\right] = 1\,, \tag{2}$$

where the probability is taken over any randomness used by the algorithms.

Security. For $\lambda, n \in \mathbb{N}$, and $i, j \in [n]$, define the distribution

$$D_{\lambda,n,i} := \left\{ q : \begin{array}{r}(\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n) \\ q \leftarrow \mathsf{Query}(\mathsf{ck}, i)\end{array} \right\}, \tag{3}$$

and for an adversary $\mathcal{A}$, define the adversary's advantage as

$$\mathsf{PIRadv}[\mathcal{A}, \Pi](\lambda, n) := \max_{i,j \in [n]} \left\{ \Pr\left[\mathcal{A}(1^\lambda, D_{\lambda,n,i}) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, D_{\lambda,n,j}) = 1\right] \right\}.$$

Scheme Π is *computationally secure* if for every polynomially bounded function $n(\lambda)$ and every efficient adversary $\mathcal{A}$, the quantity $\mathsf{PIRadv}[\mathcal{A}, \Pi](\lambda, n(\lambda))$ is a

negligible function of λ. In particular, we say it is ϵ-secure if this advantage is at most $\epsilon(\lambda, n)$. The scheme is *statistically secure* if the same holds true even for computationally unbounded adversaries.

Remark 9 (Online running time). In Definition 8, the online server's answer algorithm Answer gets oracle access to the bits of the database x. We do so to emphasize that, for all of the PIR schemes described in this paper, the online server runs in time *sublinear* in the database size n, and can thus reply to the client's query after probing only $o(n)$ bits of the database. In practice, the online server could implement each oracle call using a lookup to the database in $\widetilde{O}(1)$ time, in a reasonable model of computation (e.g., the RAM model).

Remark 10 (Information-theoretic PIR as offline/online PIR). It turns out that *any* two-server PIR scheme with perfect information-theoretic security can be cast as an offline/online PIR scheme. To see why: in a two-server perfectly secure PIR, the distribution over query strings that the client sends to each server is independent of the database bit that the client wants to read. (If not, the scheme cannot possibly be perfectly secure.) Thus, the client can query one of the two servers server before it knows which database bit it wants to read.

However, in all existing two-server perfectly secure PIR schemes, *both* servers run in time $\Omega(n)$ on databases of size n. Therefore, viewing any standard two-server PIR scheme as an offline/online scheme yields a two-server offline/online PIR scheme in which the online running time is $\Omega(n)$. In contrast, we construct offline/online PIR schemes in which the online server runs in time $o(n)$.

3.2 New constructions

The following theorem, which we prove at the end of this subsection, captures our main result on two-server offline/online PIR. It shows that it is possible to simultaneously achieve sublinear total communication and sublinear online time:

Theorem 11 (Two-server statistically secure offline/online PIR). *There exists a* statistically secure *two-server offline/online PIR scheme, such that on every n-bit database and every security parameter $\lambda \in \mathbb{N}$:*

- *the offline phase uses $O(\lambda\sqrt{n}\log^2 n)$ bits of communication,*
- *the offline server runs in time $\widetilde{O}_\lambda(n)$,*
- *the online phase uses $O(\lambda\sqrt{n}\log n)$ bits of communication,*
- *the online server runs in time $\widetilde{O}_\lambda(\sqrt{n})$, and*
- *the client uses time and memory $\widetilde{O}_\lambda(\sqrt{n})$.*

Moreover, the security advantage of any adversary is at most $\mathrm{poly}(\lambda, n) \cdot 2^{-\lambda}$.

Remark 12 (Concrete efficiency). For simplicity, we give the running times of the routines in our schemes up to $\mathrm{poly}(\lambda, \log n)$-factors. It is possible to make these hidden factors as small as $O(\lambda \log n)$.

Remark 13 (Trading communication for online time). By adjusting the parameters of the construction, it is possible to generalize Theorem 11 to give a two-server

offline/online PIR scheme in which, for any function $C\colon \mathbb{N} \to \mathbb{N}$ with $C(n) \le n/2$, the offline phases uses $C(n)$ bits of communication, and the online server runs in time $\widetilde{O}(n/C(n))$. This adjustment requires the client and preprocessing server to have access to a sequence of common random bits, or, in the computational setting, assuming the existence of pseudorandom generators.

In Appendix C.1 we discuss additional issues such as support of databases with longer rows, further reducing the client's online time via a connection to the 3-SUM problem, and implications of Theorem 11 for random self-reductions.

The following theorem, which we prove at the end of this subsection, shows that, if we settle for only computational—rather than statistical—security, we can decrease the online communication cost of the PIR scheme of Theorem 11 from $O_\lambda(\sqrt{n}\log n)$ to $O_\lambda(\log n)$ without degrading any other efficiency metrics. It also allows us to slightly decrease the offline communication cost.

Theorem 14 (Two-server computational offline/online PIR). *Assuming the existence of pseudorandom generators, there exists a two-server offline/online PIR scheme Ψ that satisfies the efficiency criteria of Theorem 11, except that*
- *the communication cost of the offline phase decreases to $O(\lambda\sqrt{n}\log n)$,*
- *the communication cost of the online phase decreases to $O(\lambda^2 \log n)$, and*
- *if the underlying PRG is ϵ_G-secure, the PIR scheme is ϵ-secure for $\epsilon(\lambda, n) = \mathrm{poly}(\lambda, n) \cdot (\epsilon_G(\lambda, n) + 2^{-\lambda})$.*

The main building block we use to construct two-server PIR schemes with low communication complexity and low online server time is puncturable pseudorandom sets with small keys. We give the construction in the next subsection, and prove the following lemma about the construction in Appendix C.2.

Lemma 15. *Let $s\colon \mathbb{N} \to \mathbb{N}$ be any function such that $s(n) \le n/2$. Let Ψ be an ϵ_Ψ-secure puncturable pseudorandom set with set size s, key size κ, and punctured key size κ_p. Then there exists a two-server ϵ-secure offline/online PIR scheme Π_Ψ, such that on security parameter λ and every n-bit database, in the offline phase:*
- *the client sends $\lambda\kappa + (\lambda n/s(n)) \log^2 n$ bits to the server,*
- *the offline server runs in time $n \cdot \mathrm{poly}(\lambda, \log n)$,*
- *the offline server's answer is $O((\lambda n/s(n)) \log n)$ bits in length.*

In the online phase:
- *the client sends $\lambda\kappa_\mathsf{p}$ bits to the server,*
- *the online server runs in time $s(n) \cdot \mathrm{poly}(\lambda, \log n)$, and*
- *the online server's answer consists of λ bits.*

Furthermore,
- *the client runs in time $(s(n) + n/s(n)) \cdot \mathrm{poly}(\lambda, \log n)$ and stores $O(\lambda\kappa + (\lambda n/s(n)) \log^2 n)$ bits between the offline and online phases, and*
- *the advantage $\epsilon(\lambda, n) \le \mathrm{poly}(\lambda, n) \cdot \left(\epsilon_\Psi(\lambda, n) + 2^{-\lambda}\right)$.*

Theorem 11 follows by instantiating Lemma 15 with the information-theoretic puncturable pseudorandom set construction of Fact 2, which has keys and puncturable keys of length at most $(s + O(1)) \log n$, and by setting $s = \sqrt{n}$.

Theorem 14 follows by instantiating Lemma 15 with the puncturable pseudorandom set of Corollary 6, which has keys of length $O(\lambda)$ and punctured keys of length $O(\lambda \log n)$, and setting $s = \sqrt{n}$. Additionally we reduce the offline communication from $O(\lambda\sqrt{n}\log^2 n)$ to $O(\lambda\sqrt{n}\log n)$ by replacing the random shifts used in Construction 16 with pseudorandom ones, generated from one seed of length λ.

Construction 16 (Two-server PIR with sublinear online time). The construction is parametrized by set size $s\colon \mathbb{N} \to \mathbb{N}$ and uses a puncturable pseudorandom set $\Psi = (\mathsf{Gen}, \mathsf{Punc}, \mathsf{Eval})$ with key space $\mathcal{K}$, punctured-key space $\mathcal{K}_\mathsf{p}$, and set size s, extended by routines $(\mathsf{Shift}, \mathsf{GenWith})$. The final scheme is obtained by running λ instances of this scheme in parallel. Throughout, let $m \coloneqq (n/s(n)) \log n$.

Offline phase

$\underline{\mathsf{Setup}(1^\lambda, n) \to \mathsf{ck}, q_h}$
$\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)$
sample $\delta_1, \ldots, \delta_m \xleftarrow{\mathrm{R}} [n]$
$\mathsf{ck} \leftarrow (\mathsf{sk}, \delta_1, \ldots, \delta_m)$
output ck and $q_h \leftarrow \mathsf{sk}$

$\underline{\mathsf{Hint}(q_h, x \in \{0,1\}^n) \to h \in \{0,1\}^m}$
parse q_h as $\mathsf{sk} \in \mathcal{K}$ and $\delta \in [n]^m$
for $j = 1, \ldots, m$ do:
  $S_j \leftarrow \mathsf{Eval}(\mathsf{Shift}(\mathsf{sk}, \delta_j))$
  $h_j \leftarrow \sum_{i \in S_j} x_i \bmod 2$
output $h \leftarrow (h_1, \ldots, h_m)$

Online phase

$\underline{\mathsf{Query}(\mathsf{ck}, i \in [n]) \to q \in \mathcal{K}_\mathsf{p}}$
parse ck as $\mathsf{sk} \in \mathcal{K}$ and $\delta \in [n]^m$
sample a bit $b \xleftarrow{\mathrm{R}} \mathsf{Bernoulli}(\frac{s-1}{n})$
find a $j \in [m]$ s.t. $i - \delta_j \in \mathsf{Eval}(\mathsf{sk})$
if such a $j \in [m]$ exists:
  $\mathsf{sk}_q \leftarrow \mathsf{Shift}(\mathsf{sk}, \delta_j)$
otherwise:
  $j \leftarrow \perp$
  $i' \xleftarrow{\mathrm{R}} \mathsf{Eval}(\mathsf{sk})$
  $\mathsf{sk}_q \leftarrow \mathsf{Shift}(\mathsf{sk}, i - i')$
if $b = 0$: $i_\mathsf{punc} \leftarrow i$
else: $i_\mathsf{punc} \xleftarrow{\mathrm{R}} \mathsf{Eval}(\mathsf{sk}_q) \setminus \{i\}$
output $q \leftarrow \mathsf{Punc}(\mathsf{sk}_q, i_\mathsf{punc})$

$\underline{\mathsf{Answer}^x(q \in \mathcal{K}_\mathsf{p}) \to a \in \{0,1\}}$
$S \leftarrow \mathsf{Eval}(q)$
return $a \leftarrow \sum_{i \in S} x_i \bmod 2$

$\underline{\mathsf{Reconstruct}(h \in \{0,1\}^m, a \in \{0,1\}) \to x_i}$
let j and b be as in $\mathsf{Query}^\dagger$
if $j = \perp$ or $b = 0$ then output $\perp$
output $x_i \leftarrow h_j - a \bmod 2$

† For simplicity, we avoid passing j and b explicitly from Query to Reconstruct.

3.3 Construction of PIR from puncturable pseudorandom sets

We first present an overview of the construction. The formal specification appears in Construction 16, and the full analysis appears in Appendix C.

The PIR scheme makes use of a puncturable pseudorandom set $\Psi = (\mathsf{Gen}, \mathsf{Punc}, \mathsf{Eval})$ with set size $s(n)$ extended by routines $(\mathsf{Shift}, \mathsf{GenWith})$. We denote $s \coloneqq s(n)$ and assume without loss of generality that $s \geq \log n$, as otherwise, a scheme in

which the offline server sends the entire database to the client trivially satisfies the lemma. We also define $m := (n/s)\log n$. The PIR scheme operates in two phases, in each of which the client interacts with one of the two servers:

Offline phase.

1. The client samples a random set key $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)$ for universe size n and set of size s. It also samples m random shifts $\delta_1, \ldots, \delta_m \in [n]$. The client sends the set key and the shifts to the offline server.
2. Upon receiving the set key sk and the random shifts $\delta_1, \ldots, \delta_m$ from the client, the offline server expands the set key to get the set $S \leftarrow \mathsf{Eval}(\mathsf{sk}) \subseteq [n]$. Each shift $\delta_j \in [n]$ defines a "shifted" set $S_j \leftarrow \{x + \delta_j \bmod n \mid x \in S\}$ (when adding elements in $[n]$, we identify it with $\mathbb{Z}_n$).
   For each shift δ_j, the offline server computes the parity of the bits pointed by the shifted set S_j, i.e., sets $h_j := \sum_{i \in S_j} x_i \bmod 2$. These bits constitute the hint $h = (h_1, \ldots, h_m) \in \{0,1\}^m$, which the server sends to the client.

Online phase. The client takes as input an index $i_{\mathsf{pir}} \in [n]$ of the database it wants to query. The client has its set key sk and the shifts vector δ from the offline phase and the hint $h \in \{0,1\}^m$ from the offline server.

1. The client expands the set key sk into the set $S \leftarrow \mathsf{Eval}(\mathsf{sk})$. It then searches for a value $j \in [m]$ such that $i_{\mathsf{pir}} + \delta_j \in S$. (The client can execute this search in $O(m+n)$ time using a hash table.)
   - If such a shift δ_j exists, the client computes the corresponding shifted set key $\mathsf{sk}_q \leftarrow \mathsf{Shift}(\mathsf{sk}, \delta_j)$, so that i_{pir} falls into the set $\mathsf{Eval}(\mathsf{sk}_q)$.
   - If such an index does not exist, the client samples an element $i \xleftarrow{\mathrm{R}} S$ and computes the shifted set $\mathsf{sk}_q \leftarrow \mathsf{Shift}(\mathsf{sk}, i_{\mathsf{pir}} - i')$.

   Either way, we refer to the chosen set key as sk_q and it holds $i_{\mathsf{pir}} \in \mathsf{Eval}(\mathsf{sk}_q)$.
2. The client samples a bit $b \xleftarrow{\mathrm{R}} \mathsf{Bernoulli}((s-1)/n)$ and then chooses an element i_{punc} at which to puncture its set key sk_q.
   - If $b = 0$, the client punctures the key sk_q at the point: $i_{\mathsf{punc}} \leftarrow i_{\mathsf{pir}}$.
   - If $b = 1$, the client punctures the key sk_q at a random point: $i_{\mathsf{punc}} \xleftarrow{\mathrm{R}} \mathsf{Eval}(\mathsf{sk}_q) \setminus \{i_{\mathsf{pir}}\}$.

   The client sends the punctured key $q \xleftarrow{\mathrm{R}} \mathsf{Punc}(\mathsf{sk}_q, i_{\mathsf{punc}})$ to the online server. (In the proof, we show that this punctured key computationally hides the index i_{pir} of the bit that the client wants to fetch from the database.)
3. The online server computes the punctured set $S^* \leftarrow \mathsf{Eval}(q) \subseteq [n]$ and views this set as $s-1$ pointers to bits in the database $x \in \{0,1\}^n$. The online server computes the parity of these $s-1$ bits: $a \leftarrow \sum_{i \in S^*} x_i \bmod 2$. The online server then returns this parity to the client. Notice that the online server only needs to probe $s-1$ bits of the database and can run in time $s \cdot \mathrm{poly}(\lambda, \log n)$.
4. If, in Step 2, the client's random bit $b = 0$, the client can recover the bit at position i_{pir} in the database from the hint h and the answer a by computing $(h - a) \bmod 2 = \sum_{i \in S} x_i - \sum_{S^*} x_i = \sum_{i \in S} x_i - \sum_{S \setminus i_{\mathsf{pir}}} x_i = x_{i_{\mathsf{pir}}}$.

Note that the scheme fails if either $i_{\mathsf{punc}} \neq i_{\mathsf{pir}}$ or $i_{\mathsf{pir}} \notin \cup_{j\in[m]} S_j$. The probability of the former is $(s-1)/n$ and, by setting $m \approx n \log n/s$, we can drive down the probability of the latter to be approximately $1/n$. By running $O(\lambda)$ instances of the scheme in parallel, using independent randomness for each instance, we can drive the overall failure probability to be negligible in λ.

It is now possible to transform the PIR scheme into one with perfect correctness, at the expense of a negligible security loss. To do so, if the client detects an error (which happens with only a negligible probability), it simply reads its desired bit from the database using a non-private lookup. (Achieving perfect correctness *and* security is also possible, at the cost of having an offline phase that runs in *expected* polynomial time.)

4 Two-server PIR with sublinear amortized time

One shortcoming of the PIR scheme of the previous section is that every execution of its offline phase supports only one subsequent query. To perform each additional query, the client and the server must rerun the offline phase. Therefore, although the online query-processing time is sublinear, the overall cost of each query, including that of the offline phase, remains linear.

We now extend the scheme of the previous section such that a single execution of the offline phase enables the client to subsequently query the database *polynomially many times*, without ever having to rerun the offline phase. The extended scheme is nearly as efficient as the basic, single-query scheme. The only loss in efficiency is the online communication, which increases to $\widetilde{O}(n^{1/2})$. We stress that the client can choose the retrieved indices adaptively, and so our scheme does *not* rely on jointly processing a batch of queries.

Our security definition, given in Appendix D, accounts for an *active* (fully malicious) adversary that controls either of the two servers, and can adaptively choose the database indices that the client queries. Here, we give our main result:

Theorem 17 (Two-server multi-query offline/online PIR). *Assuming the existence of pseudorandom permutations, there exists a two-server* multi-query *offline/online PIR scheme, such that on every n-bit database and every security parameter $\lambda \in \mathbb{N}$, in the offline phase:*

- *the offline server runs in time $\widetilde{O}_\lambda(n)$,*
- *the total communication is $O(\lambda\sqrt{n}\log n)$ bits,*

and in the online phase:

- *the online server runs in time $\widetilde{O}_\lambda(\sqrt{n})$,*
- *the total communication is $O(\lambda\sqrt{n}\log n)$ bits, and*
- *if the underlying PRP is ϵ_P-secure, the PIR scheme is ϵ-secure for $\epsilon(\lambda, n) \leq \mathrm{poly}(\lambda, n) \cdot (\epsilon_P(\lambda, n) + 2^{-\lambda})$.*

Furthermore, the client uses offline time, storage, and online time $\widetilde{O}_\lambda(n^{1/2})$.

Construction 44 fully specifies the scheme that proves Theorem 17. The full analysis appears in Appendix D, where we also prove the following corollary:

Corollary 18 (Reducing communication). *Assuming the existence of pseudorandom generators, there exists a scheme as in Theorem 17, albeit*

- *the client offline time increases to* $\widetilde{O}_\lambda(n)$,
- *the client storage and online time increases to* $\widetilde{O}_\lambda(n^{5/6})$, *and*
- *the total online communication decreases to* $O(\lambda^2 \log n)$.

Remark 19. As in Section 3, it is possible to achieve statistical security, by replacing the computationally secure puncturable pseudorandom set in the proof of Theorem 17, with a perfectly secure one and applying a standard "balancing" technique [CGKS95, Section 4.3] to get a scheme with online work and communication $\widetilde{O}_\lambda(n^{2/3})$.

4.1 Sketch of the construction

We sketch the construction here, but refer to Appendix D for the details.

Our starting point is the single-query scheme of Section 3. There, the hint consists of a list of $m = \sqrt{n} \log n$ random sets $S_1, \ldots, S_m \subseteq [n]$, each of size roughly $\sqrt{n}$, represented by m puncturable pseudorandom set keys, along with the parity of the database bits in each set. In the online phase, to read the ith database bit, the client finds a set $S_j \in \{S_1, \ldots, S_m\}$ such that $i \in S_j$ and with good probability sends to the right server the set $S' = S_j \setminus \{i\}$. Once the client has used the set S_j to make a query, the client cannot use S_j again. If the client used S_j to query for another index i', the right server would, with good probability, see $S_j \setminus \{i\}$ and $S_j \setminus \{i'\}$. Taking the difference of these sets would reveal the secret indices $\{i, i'\}$ to the right server, breaking security.

The key to supporting multiple queries with only one execution of the offline phase is to have the client "refresh" its hint every time it queries the database. We refer to the two servers as "left" and "right". The left server provides the hint to the client in the offline phase, and later helps the client to refresh that hint after each subsequent read operation. The right server answers the queries that allow the client to reconstruct the database bits it is attempting to read (as in our constructions of Section 3).

The online-phase interaction with the right server proceeds exactly as in the single-query scheme: the client sends a punctured set to the right server and recovers the bit x_i. However, the client in the multi-query scheme must somehow replace the set S_j (and the corresponding parity bit) with a fresh random set S_{new}. To make this work, we must answer two questions: (i) How does the client sample the set S_{new}? and (ii) How does the client fetch the corresponding parity bit $\sum_{i \in S_{\mathsf{new}}} x_i \bmod 2$?

First, for correctness and privacy to hold for future queries, the client must sample the replacement set S_{new} in a way that preserves the joint distribution of the sets $S_1, \ldots, S_m$. Notice that sampling a fresh random set S_{new} of the proper size will *not* work, since it distorts the joint distribution of the sets. In particular, replacing a set S_j that contains i with a fresh random set causes the expected number of sets in $S_1, \ldots, S_m$ containing i to decrease. What *does* work is to

have the client sample a fresh random set S_{new} subject to the constraint that it contains the index i that the client just read. This is possible since, as described in Section 2.3, punctured sets support biased sampling.

Second, the client needs to construct the correct parity bit $h_{\mathsf{new}} = \sum_{i \in S_{\mathsf{new}}} x_i \bmod 2$ for the new set S_{new}. The client obtains the new parity bit by (1) puncturing the set S_{new} at element i and (2) querying the left server on the punctured set. The left server then replies with the parity of the bits in the punctured set $S_{\mathsf{new}} \setminus \{i\}$. At this point the client can recover the parity of the new set S_{new} by adding the reply from the left server and the value x_i, which it reconstructs, as in the single-query case, using the reply from the right server.

The final complication is that, as in Section 5, in order for the punctured set to look random, the client occasionally needs to send to the servers a set punctured at the retrieved index i. In this case, the read operation fails. When this happens, the client sends a punctured version of the new set S_{new} to both servers, the client leaves its hint state unchanged, and the read operations fails.

As in Section 5, by running λ instances of the scheme in parallel we can drive the overall failure probability to be negligible in λ. We can then trade the failure probability for a negligible security loss and get a perfectly correct scheme.

5 Single-server PIR with sublinear online time

In this section, we introduce *single-server* offline/online PIR. The syntax and correctness properties of a single-server offline/online PIR scheme, formally defined in Definition 46 in Appendix E, are exactly as in Definition 8. The key difference is that, in the single-server setting, the client interacts with the *same* server in both the offline phase and the online phase. Still the server should learn nothing about the database index the client wants to retrieve.

Unlike in the two-server setting, where we can achieve statistical security, in the single-server setting, we must rely on computational assumptions [CGKS95]. Since non-trivial single-server PIR implies oblivious transfer [DMO00], our assumptions must imply public-key cryptography.

Our single-server schemes shift all of the expensive work of responding to the client's PIR query—the linear-time scan over the database and the public-key operations—into the offline phase. The server can then respond to the client's query in the online phase much more quickly, with

- ***no*** public-key cryptographic operations and
- server time ***sublinear*** in the size of the database.

Our main construction (Theorem 20) achieves $\widetilde{O}_\lambda(n^{2/3})$ communication and online time and $\widetilde{O}_\lambda(n)$ server computational time in the offline phase, using linearly homomorphic encryption and standard single-server PIR. We also sketch an asymptotically superior construction (Theorem 22) that achieves $\widetilde{O}_\lambda(n^{1/2})$ communication and online time, at the cost of using fully homomorphic encryption [Gen09]. Our lower bound of Section 6 proves the optimality of this latter scheme, up to log factors, with respect to the trade-off between offline

communication and online time, given the restriction that the server must store the database in unencoded from and use no extra storage.

A drawback of our single-server PIR schemes is that they have polynomial communication $\Omega(n^{1/2})$, which is higher than the polylog(n) communication of state-of-the-art standard single-server PIR schemes [CMS99]. That said, in some applications, the benefits of sublinear online time and no public-key cryptography in the online phase may outweigh the costs.

The main result of this section is:

Theorem 20 (Single-server offline/online PIR). *Suppose there exist:*
- *a linearly homomorphic encryption scheme (Appendix A.3) with ciphertext space $\mathbb{G}$ and*
- *single-server PIR with communication cost* $\mathrm{poly}(\lambda, \log n)$ *and server computation time $\widetilde{O}_\lambda(n)$ (for every database size n and security parameter $\lambda \in \mathbb{N}$).*

Then, there exists a single-server offline/online PIR scheme, that makes black-box use of the group $\mathbb{G}$, such that for every security parameter $\lambda \in \mathbb{N}$ and n-bit database, it uses
- *in the offline phase: $\widetilde{O}_\lambda(n^{2/3})$ bits of communication and $\widetilde{O}_\lambda(n)$ operations in $\mathbb{G}$, and*
- *in the online phase: $\widetilde{O}_\lambda(n^{1/3})$ bits of communication, $\widetilde{O}_\lambda(n^{2/3})$ time, and no operations in $\mathbb{G}$.*

Moreover, the client uses time and memory $\widetilde{O}_\lambda(n^{2/3})$.

We prove Theorem 20 in Section 5.1.

Remark 21 (A much simpler scheme). In Appendix E.2, we give a very simple—and likely easy-to-implement—single-server offline/online scheme that requires only linearly homomorphic encryption and has $O(\sqrt{n})$ total communication, online time, and client storage. The scheme uses no public-key cryptographic operations in the online phase, and its simplicity makes it potentially attractive for practical applications. The downside is that its online phase requires a *linear* number of bit operations (but no public-key operations).

Patel, Persiano, and Yeo [PPY18] give an offline/online scheme with linear communication and linear online server time (but a sublinear number of online public-key operations) while this simple scheme has sublinear communication and no public-key operations in the online phase. In contrast, the client in their scheme can use a single offline phase for many online operations, while our single-server scheme requires an offline phase before each online query.

5.1 Proof of Theorem 20

We first construct a single-server PIR scheme with linear client upload in the offline phase. We then trade upload for download to complete the proof.

An unbalanced scheme. The main idea is to have one server do the work of *both* servers in the two-server offline/online PIR construction of Theorem 14. For the construction to remain secure, the offline phase in the single-server scheme

must not leak to the server anything about the offline query of the underlying two-server scheme. If we achieve this property, the client would be safe to run both phases of the two-server construction with a single server:

- In the offline phase, the client obtains the hint without leaking anything about it to the server.
- The online phase proceeds exactly as in the two-server scheme, except that the client interacts with the same server in both phases.

Our starting point is the two-server offline/online PIR construction of Theorem 14. There, the client sends to the offline server a set $S \subseteq [n]$, represented as a puncturable pseudorandom set key, and shift values $(\delta_1, \ldots, \delta_m) \in [m]$, where $m = \sqrt{n} \log n$. For each shift δ_j, the server returns the parity of the database bits in the set S, shifted by δ_j. That is, the client obtains $(h_{\delta_1}, \ldots, h_{\delta_m})$ for $h_{\delta_j} \leftarrow \sum_{i \in S} x_{i+\delta_j} \bmod 2$.

We modify the offline phase such that the client can retrieve the bits $(h_{\delta_1}, \ldots, h_{\delta_m})$ from the server without revealing to it neither the set S nor the shifts $(\delta_1, \ldots, \delta_m)$.

- **Client sends an encryption of its set S to the server.** The client generates encryption key $k \leftarrow \mathsf{Gen}(1^\lambda)$ for the linearly homomorphic encryption scheme assumed in the theorem. The client then computes a vector $v \in \{0,1\}^n \subseteq \mathbb{F}^n$, where $v_i = 1$ if and only if $i \in S$ (i.e., v is the indicator vector of S embedded into $\mathbb{F}^n$). The client encrypts v component by component using the linearly homomorphic encryption scheme Enc to get a ciphertext $\mathsf{ct}_S = \mathsf{Enc}_k(v) \in \mathbb{G}^n$. The client sends this ciphertext to the server.
- **Server computes the parities for all possible shifts of S.** Given the ciphertext ct_S, the server computes the component-wise encryption ct_h of the vector $h \in \mathbb{F}^n$ where $h_j \leftarrow \sum_{i \in S} x_{i+j} \in \mathbb{F}$ for $j \in [n]$.

  We observe that the vector $h \in \mathbb{F}^n$ can be expressed as $h = v \cdot \mathsf{Circ}(x)$, where $\mathsf{Circ}(x) \in \mathbb{F}^{n \times n}$ is the circulant matrix defined by the vector $x \in \mathbb{F}^n$. A naïve method for computing this product would require $\widetilde{\Omega}_\lambda(n^2)$ time at the server. Instead, the server can compute the vector-matrix product $\mathsf{ct}_h = \mathsf{ct}_S \cdot \mathsf{Circ}(x)$ using an FFT-like computation in only $\widetilde{O}_\lambda(n)$ time. (To allow an FFT computation, the plaintext space of the linearly homomorphic encryption scheme must be a field that has a primitive n-th root of unity. We elide this technical restriction, since it is easy to satisfy in practice.)
- **Client retrieves the encrypted parities using PIR.** At this point, the server holds a ciphertext vector $\mathsf{ct}_h \in \mathbb{G}^n$ which consists of the encryptions of the parities for all n possible shifts. The client holds shifts $(\delta_1, \ldots, \delta_m) \in [n]$ and wants to retrieve values $h_{\delta_1}, \ldots, h_{\delta_m}$ from the server, without revealing which values it is fetching.

  To this end, the client uses PIR to retrieve its desired components from the server. A standard single-server PIR scheme with polylogarithmic communication and linear server time (as assumed in the statement of the theorem) allows retrieving each of these components using $\mathrm{poly}(\lambda, \log n)$ communication and $\widetilde{O}_\lambda(n)$ server work. Implemented in a straightforward way, retrieving m components would increase the server work m-fold. However, the batch-

PIR technique of Ishai, Kushilevitz, Ostrovsky, and Sahai [IKOS04, Section 5] allows the server to respond to a batch of m queries with $m \cdot \mathrm{poly}(\lambda, \log n)$ bits of communication and only $\widetilde{O}_\lambda(n)$ work overall.

– **Client recovers the hint.** Finally, the client decrypts the m ciphertexts to recover the values $h_{\delta_1}, \ldots, h_{\delta_m}$. Note that $h_{\delta_j} = \sum_{i \in S} x_{i+\delta_j} \in \mathbb{F}$, and whenever $|\mathbb{F}| > n$ the sum also holds over the integers. Thus reducing h_{δ_j} modulo 2 yields the jth bit of the hint.

Given the hint, the client can subsequently execute the exact same online phase as in the two-party scheme. However, as the server has learned nothing about the hint request, the client can run the online phase with the same server.

The correctness of the scheme follows immediately from the correctness of the underlying single-server PIR and two-server offline/online PIR. We sketch the security argument in Appendix E.

Efficiency. In the offline phase, the client uploads n ciphertexts, at a communication cost of $\widetilde{O}_\lambda(n)$ bits, and then performs m single-server PIR queries, at cost $m \cdot \mathrm{poly}(\lambda, \log n)$ bits, for $m = \sqrt{n} \log n$. So the total communication cost is $\widetilde{O}_\lambda(n)$ bits. The offline server computation consists of computing the matrix-vector product, at cost $\widetilde{O}_\lambda(n)$ using an FFT, and replying to the batch PIR query using $\widetilde{O}_\lambda(n)$ work. The total offline work of the server is therefore $\widetilde{O}_\lambda(n)$. In the online phase, the server does $\widetilde{O}_\lambda(\sqrt{n})$ work, and the communication is $O_\lambda(\log n)$, exactly as in the protocol of Theorem 14. The client's running time and storage are also as in Theorem 14.

Rebalancing the scheme. To complete the proof of Theorem 20, we now use a standard "balancing" idea from the PIR literature [CGKS95, Section 4.3], that allows us to trade client upload for client download.

The client and server partition the n-bit database into $n^{1/3}$ buckets, each of size $n' = n^{2/3}$ bits, and then run the above unbalanced scheme $n^{1/3}$ times—once using each buckets. The only change is that the client sends a *single* query to the server (rather than one query per bucket) in both the offline and online phases. The resulting scheme has total communication complexity $\widetilde{O}_\lambda(n') + n^{1/3} \cdot \widetilde{O}_\lambda(\sqrt{n'}) = \widetilde{O}_\lambda(n^{2/3})$ bits, and online work $n^{1/3} \cdot \widetilde{O}_\lambda(\sqrt{n'}) = \widetilde{O}_\lambda(n^{2/3})$ for both the server and the client.

Improving efficiency with higher-order homomorphisms.

If we use a homomorphic encryption scheme that supports degree-two [BGN05] or higher-degree homomorphic computation, we can build offline/online PIR schemes that provide even better communication efficiency. For example, given a fully homomorphic encryption scheme [Gen09] (FHE), we can use the idea of Theorem 20 with the two-server PIR scheme of Construction 16 to obtain:

Theorem 22 (Informal). *Assume fully homomorphic encryption exists. Then, for all security parameters $\lambda \in \mathbb{N}$, there is a single-server offline/online PIR*

scheme on n-bit databases that uses $\widetilde{O}_\lambda(\sqrt{n})$ bits of communication and $\widetilde{O}_\lambda(\sqrt{n})$ server-side time in the online phase.

The observation is that, in the two-server setting (Construction 16), the client only sends the server a PRG seed. By using FHE, the client in the single-server setting could send the server an encryption of that seed, and the server could homomorphically evaluate the offline server's algorithm on the encrypted seed. The online phase remains the same. In Appendix E.3, we discuss possible routes towards obtaining a similarly efficient scheme under weaker assumptions.

6 Lower bound for PIR with sublinear online time

In this section, we prove that the offline/online PIR schemes we construct in Section 3 achieve the optimal trade-off, up to log factors, between

- the number of bits C that the client downloads in the offline phase and
- the running time T of the server in the online phase.

Specifically, we show that any offline/online PIR scheme, in which the servers store the database in its unmodified form and use no additional storage, and that succeeds with constant probability on a database of size n, must have $(C+1)(T+1) = \widetilde{\Omega}(n)$.

The fact that we are able to obtain a polynomial lower bound on the communication complexity of offline/online PIR schemes may be somewhat surprising, as it has been notoriously difficult to obtain communication lower bounds for standard two-server PIR, in which the servers' running time is unbounded. In particular, in the information-theoretic setting, the best communication lower bound for two-server PIR stands at $C \geq (5 - o(1)) \cdot \log_2 n$ bits. In contrast, for two-server PIR schemes in which one of the servers is restricted to run in time $T \leq \sqrt{n}$, we obtain a polynomial communication lower bound of $C \geq \widetilde{\Omega}(\sqrt{n})$.

Our lower bound holds even against offline/online PIR schemes that provide only *computational security*, as well as against *single-server* offline/online PIR schemes. Our PIR schemes of Section 3 achieve this bound, up to logarithmic factors, as does the single-server scheme of Theorem 22.

Theorem 23. *Consider a computationally secure offline/online PIR scheme such that, on security parameter $\lambda \in \mathbb{N}$ and database size $n \in \mathbb{N}$,*

- *the client downloads C bits in the offline phase,*
- *the online server stores the database in its original form and probes T bits of the database in the course of processing the client's query, and*
- *the client recovers its desired bit with probability at least ϵ, over the choice of its randomness.*

Then, for polynomially bounded $n = n(\lambda)$, it holds that

$$\epsilon \leq 1/2 + \widetilde{O}\left(T/n + \sqrt{C(T+1)/n}\right) + \mathrm{negl}(\lambda),$$

and in particular for $\epsilon \geq 1/2 + \Omega(1)$ and large enough λ it holds that

$$(C+1) \cdot (T+1) \geq \widetilde{\Omega}(n).$$

We prove Theorem 23 by showing that an offline/online PIR scheme implies a solution for a computational task called "Yao's Box Problem." Using a preexisting lower bound for the Box Problem immediately gives a communication-time lower bound on offline/online PIR schemes. The details appear in Appendix F.

Remark 24. The lower bound of Theorem 23 does not preclude schemes that achieve better communication and lower bound by virtue of having the servers store some form of encoding of the database. We discuss schemes of this form [DIO01, BIM04] in Section 1.4. In particular, constructing PIR schemes with preprocessing [BIM04] that beat the above lower bound (in terms of their communication and online time) seems like an interesting open problem.

7 Open questions

This work leaves open a number of questions:

- Is it possible to construct offline/online PIR schemes in which the client runs in total time $o(n)$, stores $o(n)$ bits, and has online running time polylog(n)?
- Does Theorem 22 follow from an assumption weaker than FHE?
- Can we construct a multi-query scheme (Section 4) with only one server?
- In Appendix G, we show how to view our PIR construction via a new abstraction that we call *sparse distributed point functions* ("sparse DPFs"), inspired by the standard notion of DPFs [GI14]. Are there even simpler constructions of sparse DPFs than the ones implied by our PIR schemes?

Acknowledgements. We gratefully acknowledge Dan Boneh for his advice on technical questions and for supporting our work on this project from the beginning. We thank Yuval Ishai for answering our questions about PIR, Sam Kim and David Wu for feedback on early versions of this work, and Helger Lipmaa for kindly pointing us to related work. Finally, we would like to thank the anonymous Eurocrypt reviewers for their many constructive comments. This work was supported by CISPA, DARPA, NSF, ONR, and the Simons Foundation.

References

ACLS18. S. Angel, H. Chen, K. Laine, and S. T. V. Setty. PIR with compressed queries and amortized query processing. *S&P* 2018.

AFK89. M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. *J. Comput. Syst. Sci.*, 39(1):21–50, 1989.

Amb97. A. Ambainis. Upper bound on communication complexity of private information retrieval. *ICALP* 1997.

AMBFK16. C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian. XPIR: Private information retrieval for everyone. *PETS*, 2016(2):155–174, 2016.

AS16. S. Angel and S. Setty. Unobservable communication over fully untrusted infrastructure. *SOSP* 2016.

BDOZ11. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. *EUROCRYPT* 2011.

BGI14. E. Boyle, S. Goldwasser, and I. Ivan. Functional signatures and pseudorandom functions. *PKC* 2014.

BGI15. E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. *EUROCRYPT* 2015.

BGI16. E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. *CCS* 2016.

BGN05. D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. *TCC* 2005.

BI01. A. Beimel and Y. Ishai. Information-theoretic private information retrieval: A unified construction. *ICALP* 2001.

BIKR02. A. Beimel, Y. Ishai, E. Kushilevitz, and J. Raymond. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. *FOCS* 2002.

BIM04. A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *J. Cryptol.*, 17(2):125–151, 2004.

BIPW17. E. Boyle, Y. Ishai, R. Pass, and M. Wootters. Can we access a database both locally and privately? *TCC* 2017.

BKM17. D. Boneh, S. Kim, and H. W. Montgomery. Private puncturable PRFs from standard lattice assumptions. *EUROCRYPT* 2017.

BLW17. D. Boneh, K. Lewi, and D. J. Wu. Constraining pseudorandom functions privately. *PKC* 2017.

BTVW17. Z. Brakerski, R. Tsabary, V. Vaikuntanathan, and H. Wee. Private constrained PRFs (and more) from LWE. *TCC* 2017.

BW13. D. Boneh and B. Waters. Constrained pseudorandom functions and their applications. *ASIACRYPT* 2013.

CC17. R. Canetti and Y. Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. *EUROCRYPT* 2017.

CDGS18. S. Coretti, Y. Dodis, S. Guo, and J. P. Steinberger. Random oracles and non-uniformity. *EUROCRYPT* 2018.

CG97. B. Chor and N. Gilboa. Computationally private information retrieval. *STOC* 1997.

CGKS95. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *FOCS* 1995.

CGKS98. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–982, 1998.

CHR17. R. Canetti, J. Holmgren, and S. Richelson. Towards doubly efficient private information retrieval. *TCC* 2017.

CMS99. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *EUROCRYPT* 1999.

DG16. Z. Dvir and S. Gopi. 2-server PIR with subpolynomial communication. *J. ACM*, 63(4):39:1–39:15, 2016.

DGI+19. N. Döttling, S. Garg, Y. Ishai, G. Malavolta, T. Mour, and R. Ostrovsky. Trapdoor hash functions and their applications. *CRYPTO* 2019.

DGK17. Y. Dodis, S. Guo, and J. Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. *EUROCRYPT* 2017.

DHS14. D. Demmler, A. Herzberg, and T. Schneider. RAID-PIR: practical multi-server PIR. *CCSW* 2014.

DIO98. G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for database private information retrieval. *PODC* 1998.

DIO01. G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for private information retrieval. *J. Cryptol.*, 14(1):37–74, 2001.

DJ01. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. *PKC* 2001.

DMO00. G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single database private information retrieval implies oblivious transfer. *EUROCRYPT* 2000.

DPSZ12. I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. *CRYPTO* 2012.

DTT10. A. De, L. Trevisan, and M. Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. *CRYPTO* 2010.

Efr12. K. Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012.

FF93. J. Feigenbaum and L. Fortnow. Random-self-reducibility of complete sets. *SIAM J. Comput*, 22(5):994–1005, 1993.

FN00. A. Fiat and M. Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput*, 29(3):790–803, 2000.

GCM+16. T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish. Scalable and private media consumption with Popcorn. *NSDI* 2016.

GDL+14. I. Goldberg, C. Devet, W. Lueks, A. Yang, P. Hendry, and R. Henry. Percy++, version 1.0, 2014. http://percy.sourceforge.net/.

Gen09. C. Gentry. A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University, 2009.

GGH+19. A. Golovnev, S. Guo, T. Horel, S. Park, and V. Vaikuntanathan. 3SUM with preprocessing: Algorithms, lower bounds and cryptographic applications, 2019. arXiv:1907.08355 [cs.DS].

GGM86. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

GI14. N. Gilboa and Y. Ishai. Distributed point functions and their applications. *EUROCRYPT* 2014.

GKLP17. I. Goldstein, T. Kopelowitz, M. Lewenstein, and E. Porat. Conditional lower bounds for space/time tradeoffs. *WADS* 2017.

GR05. C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. *ICALP* 2005.

GT00. R. Gennaro and L. Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. *FOCS* 2000.

Hel80. M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980.

Hen16. R. Henry. Polynomial batch codes for efficient IT-PIR. *PoPETs*, 2016(4):202–218, 2016.

HKW15. S. Hohenberger, V. Koppula, and B. Waters. Adaptively secure puncturable pseudorandom functions in the standard model. *ASIACRYPT* 2015.

HOWW18. A. Hamlin, R. Ostrovsky, M. Weiss, and D. Wichs. Private anonymous data access. Cryptology ePrint Archive, Report 2018/363, 2018.

IKOS04. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch codes and their applications. *STOC* 2004.

IKOS06. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography from anonymity. *FOCS* 2006.

IP07. Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. *TCC* 2007.

Ish19. Y. Ishai. Private communication, 2019.

Jue01. A. Juels. Targeted advertising... and privacy too. *CT-RSA* 2001.

KLDF16. A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle: Efficient communication system with strong anonymity. *PoPETs*, 2016(2):115–134, 2016.

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

KO00. E. Kushilevitz and R. Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. *EUROCRYPT* 2000.

KPTZ13. A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. Delegatable pseudorandom functions and applications. *CCS* 2013.

LG15. W. Lueks and I. Goldberg. Sublinear scaling for multi-client private information retrieval. *FC* 2015.

Lip05. H. Lipmaa. An oblivious transfer protocol with log-squared communication. *ISC* 2005.

Lip09. H. Lipmaa. First CPIR protocol with data-dependent computation. *ICISC* 2009.

MOT+11. P. Mittal, F. G. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. *USENIX Security* 2011.

OS07. R. Ostrovsky and W. E. Skeith. A survey of single-database private information retrieval: Techniques and applications. *PKC* 2007.

OU98. T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. *EUROCRYPT* 1998.

Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. *EUROCRYPT* 1999.

PPY18. S. Patel, G. Persiano, and K. Yeo. Private stateful information retrieval. *CCS* 2018.

PR93. P. Pudlák and V. Rödl. Modified ranks of tensors and the size of circuits. *STOC* 1993.

PRS97. P. Pudlák, V. Rödl, and J. Sgall. Boolean circuits, tensor ranks, and communication complexity. *SIAM J. Comput*, 26(3):605–633, 1997.

SCM05. L. Sassaman, B. Cohen, and N. Mathewson. The Pynchon Gate: A secure method of pseudonymous mail retrieval. *WPES* 2005.

SW14. A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. *STOC* 2014.

TDG16. R. R. Toledo, G. Danezis, and I. Goldberg. Lower-cost ϵ-private information retrieval. *PoPETs*, 2016(4):184–201, 2016.

Unr07. D. Unruh. Random oracles and auxiliary input. *CRYPTO* 2007.

Yao90. A. C.-C. Yao. Coherent functions and program checkers. *STOC* 1990.

Yek08. S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. *J. ACM*, 55(1):1:1–1:16, 2008.

A Standard definitions

A.1 Computationally indistinguishability

In our analysis of puncturable pseudorandom sets, we will need the following standard facts about computationally indistinguishable ensembles.

Let D_0 and D_1 be two distributions defined over a set X. For an algorithm $\mathcal{A}$ that takes in an element $x \in X$ and outputs a bit, we define

$$\mathsf{DistAdv}[\mathcal{A}, D_0, D_1] = \left|\Pr[\mathcal{A}(x) = 1 : x \xleftarrow{R} D_0] - \Pr[\mathcal{A}(x) = 1 : x \xleftarrow{R} D_1]\right| .$$

Let $\mathcal{D}_0 = \{D_{0,\lambda}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_1 = \{D_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ be two probability ensembles defined over a common sequence of sets $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$. We say that the two ensembles $\mathcal{D}_0$ and $\mathcal{D}_1$ are *computationally indistinguishable* if for every probabilistic polynomial time non-uniform algorithm $\mathcal{A}$, that takes as input an element $x \in X$, the quantity $\mathsf{DistAdv}[\mathcal{A}, D_{0,\lambda}, D_{1,\lambda}](\lambda)$ is negligible in λ.

Informally, the following standard fact states that if an adversary $\mathcal{A}$ can efficiently distinguish samples from some pair of distributions $D_i, D_j \in \{D_1, D_2, \ldots, D_n\}$, then there exists an efficient *non-uniform* adversary $\mathcal{B}$ that, when fed samples from a random distribution $D_i \xleftarrow{R} \{D_1, \ldots, D_n\}$, can guess i with non-trivial probability.

Proposition 25. *Let $D_1, D_2, \ldots, D_n$ be distributions over some set X, and let $\mathcal{A}$ be an algorithm that takes as input an element $x \in X$ and outputs a bit. Then, there exists an algorithm $\mathcal{B}$ that takes as input $x \in X$, and outputs an integer $i \in [n]$ such that*

$$\Pr\left[\mathcal{B}(x) = i \ : \ i \xleftarrow{R} [n], x \xleftarrow{R} D_i\right] = \frac{1}{n} + \frac{2}{n^2} \cdot \max_{i,j\in[n]} \mathsf{DistAdv}[\mathcal{A}, D_i, D_j] .$$

Moreover, the running time of $\mathcal{B}$ is linear in $\log n$ and in the running time of $\mathcal{A}$.

The implication in the other direction also holds:

Proposition 26. *Let $D_1, D_2, \ldots, D_n$ be distributions over some set X, and let $\mathcal{A}$ be an algorithm that takes as input $x \in X$, and outputs an integer $i \in [n]$. Then, there exist $i, j \in [n]$ and an algorithm $\mathcal{B}$, that takes as input an element $x \in X$ and outputs a bit, such that*

$$\Pr\left[\mathcal{A}(x) = i \ : \ i \xleftarrow{R} [n], x \xleftarrow{R} D_i\right] = \frac{1}{n} + \frac{n-1}{n} \cdot \mathsf{DistAdv}[\mathcal{B}, D_i, D_j] .$$

Moreover, the running time of $\mathcal{B}$ is linear in $\log n$ and in the running time of $\mathcal{A}$.

For a pseudorandom generator G with stretch $|G(x)| = \ell(|x|)$, an algorithm $\mathcal{A}$, and $\lambda \in \mathbb{N}$, we define the distinguishing advantage

$$\mathsf{PRGAdv}[\mathcal{A}, G](\lambda) := \mathsf{DistAdv}[\mathcal{A}, G(U_\lambda), U_{\ell(\lambda)}] ,$$

where, for $n \in \mathbb{N}$, U_n is the uniform distribution on n-bit strings.

A.2 Puncturable pseudorandom functions

Informally, a puncturable PRF [BW13, KPTZ13, BGI14, SW14, HKW15] is a PRF with an additional puncturing routine, that takes as input a PRF key k and an input x^* and outputs a "punctured" key k_p. The original key k and punctured key k_p behave identically, except that (1) it is not possible to evaluate the PRF at point x^* using the punctured key k_p and moreover (2) the punctured key "leaks nothing" about the value of the PRF at the point x^*.

A puncturable pseudorandom function family is defined over a key space $\mathcal{K}$ and a punctured-key space $\mathcal{K}_\mathsf{p}$, and is a triple of efficient algorithms (PRFGen, $\mathsf{PRFPunc}, \mathsf{PRFEval}$):

- $\mathsf{PRFGen}(1^\lambda, n) \to \mathsf{k}$, a randomized algorithm that takes as input a security parameter $\lambda \in \mathbb{N}$, expressed in unary, and an input-space size parameter $n \in \mathbb{N}$, expressed in binary, and outputs a key $\mathsf{k} \in \mathcal{K}$,
- $\mathsf{PRFPunc}(\mathsf{k}, x^*) \to \mathsf{k}_\mathsf{p}$, a deterministic algorithm that takes as input a secret key $\mathsf{k} \in \mathcal{K}$ and an input $x^* \in [n]$, and outputs a punctured key $\mathsf{k}_\mathsf{p} \in \mathcal{K}_\mathsf{p}$, and
- $\mathsf{PRFEval}(\mathsf{k}, x) \to y$, a deterministic algorithm that takes as input a key $\mathsf{k} \in \mathcal{K} \cup \mathcal{K}_\mathsf{p}$ and outputs a value $y \in [n]$.

Furthermore, a puncturable PRF must satisfy the following two notions.

Correctness. Informally, if k is a key and k_p is the key punctured at the point $x^* \in [n]$, the functions $\mathsf{PRFEval}(\mathsf{k}, \cdot)$ and $\mathsf{PRFEval}(\mathsf{k}_\mathsf{p}, \cdot)$ should be equal everywhere, except at the point x^*. Formally, for every $\lambda, n \in \mathbb{N}$, every $\mathsf{k} \leftarrow \mathsf{PRFGen}(1^\lambda, n)$, and every $x, x^* \in [n]$ such that $x \neq x^*$, it holds that

$$\mathsf{PRFEval}(\mathsf{k}, x) = \mathsf{PRFEval}(\mathsf{PRFPunc}(\mathsf{k}, x^*), x).$$

Security. Informally, a secret key punctured at a point $x^* \in [n]$ should leak nothing about the value of the PRF at the point x^*. Formally, for a security parameter $\lambda \in \mathbb{N}$, domain size $n \in \mathbb{N}$, and an adversary $\mathcal{A}$, let $W_{b,\lambda,n}$ be the event that $\mathcal{A}$ outputs "1" in Game 27 with bit $b \in \{0, 1\}$. We define the advantage of adversary $\mathcal{A}$ at attacking a puncturable PRF $\mathcal{F}$ as

$$\mathsf{pPRFAdv}[\mathcal{A}, \mathcal{F}](\lambda, n) := |\Pr[W_{\lambda,n,0}] - \Pr[W_{\lambda,n,1}]|.$$

We say that a puncturable PRF $\mathcal{F}$ is ϵ-secure if, for every $\lambda \in \mathbb{N}$, every $n \in \mathbb{N}$, and every efficient adversary $\mathcal{A}$, $\mathsf{pPRFAdv}[\mathcal{A}, \mathcal{F}](\lambda, n) \leq \epsilon(\lambda, n)$.

Remark 28. In general, punctured PRFs can have an exponentially large domain and codomain. For our applications however, we use punctured PRFs whose domain and codomain have size $n \leq \mathrm{poly}(\lambda)$, on security parameter λ.

Remark 29. Standard definitions of puncturable PRFs allow the adversary to puncture the PRF key at a *set* of points (i.e., $\mathsf{PRFPunc}$ takes a set $S \subseteq [n]$ as input). We give only the simpler definition, in which the adversary can puncture the key at a single point, since it suffices for our applications.

Game 27 (Puncturable PRF indistinguishability). For a puncturable PRF $(\mathsf{PRFGen}, \mathsf{PRFPunc}, \mathsf{PRFEval})$, security parameter $\lambda \in \mathbb{N}$, input-space size $n \in \mathbb{N}$, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the following game:

- The adversary takes as input 1^λ and n, and sends a point $x^* \in [n]$ to the challenger.
- The challenger computes:
  - $\mathsf{k} \leftarrow \mathsf{PRFGen}(1^\lambda, n)$,
  - $\mathsf{k_p} \leftarrow \mathsf{PRFPunc}(\mathsf{k}, x^*)$,
  - $y_0 \leftarrow \mathsf{PRFEval}(\mathsf{k}, x^*)$,
  - $y_1 \xleftarrow{R} [n]$,

  and sends $(\mathsf{k_p}, y_b)$ to the adversary.
- The adversary outputs a value $b' \in \{0,1\}$.

The standard definition of puncturable PRFs requires that the value at the punctured point be indistinguishable from random, even given the punctured key. We need the following straightforward lemma, which says that the value of the puncturable PRF at the punctured point is also *unpredictable*.

To this end, let $W_{\lambda,n}$ be the probability that the adversary wins in Game 30, with security parameter λ and input-space size n.

Game 30 (Puncturable PRF unpredictability). For a given puncturable PRF $(\mathsf{PRFGen}, \mathsf{PRFPunc}, \mathsf{PRFEval})$, security parameter $\lambda \in \mathbb{N}$, input-space size $n \in \mathbb{N}$, and an adversary $\mathcal{A}$, we define the following game:

- The adversary takes as input 1^λ and n, and sends a point $x^* \in [n]$ to the challenger.
- The challenger computes:
  - $\mathsf{k} \leftarrow \mathsf{PRFGen}(1^\lambda, n)$,
  - $\mathsf{k_p} \leftarrow \mathsf{PRFPunc}(\mathsf{k}, x^*)$,
  - $y^* \leftarrow \mathsf{PRFEval}(\mathsf{k}, x^*)$,

  and sends $\mathsf{k_p}$ to the adversary.
- The adversary outputs a value $y' \in [n]$.

We say that the adversary "wins" if $y' = y^*$.

We define the unpredictability advantage of an adversary $\mathcal{A}$ at attacking a punctured PRF $\mathcal{F}$ as

$$\mathsf{pPRFAdv}^{\mathrm{unp}}[\mathcal{A}, \mathcal{F}](\lambda, n) = \left| W_{\lambda,n} - \frac{1}{n} \right|.$$

Lemma 31. *For every algorithm $\mathcal{A}$ attacking a puncturable PRF $\mathcal{F}$ in the unpredictability setting, there is an algorithm $\mathcal{B}$ attacking $\mathcal{F}$ in the indistinguishability*

setting such that for all $\lambda, n \in \mathbb{N}$,

$$\mathsf{pPRFAdv}^{\mathrm{unp}}[\mathcal{A}, \mathcal{F}](\lambda, n) = \mathsf{pPRFAdv}[\mathcal{B}, \mathcal{F}](\lambda, n).$$

Moreover, $\mathcal{A}$ *and* $\mathcal{B}$ *have the same asymptotic running time.*

Proof. We first construct the algorithm $\mathcal{B}$ and then argue for its correctness. The algorithm $\mathcal{B}$ operates as follows:

- Run $\mathcal{A}$ to get the value x^* and send this to the challenger.
- Receive the punctured secret key $\mathsf{k_p}$ from the challenger, along with y_b.
- Run $\mathcal{A}$ on the punctured secret key $\mathsf{k_p}$ to get a value y'.
  - If $y' = y_b$, output "1."
  - Otherwise, output "0."

When running $\mathcal{B}$ in the indistinguishability game, the view of the underlying $\mathcal{A}$ is exactly as in the unpredictability game. Recalling that $y_0 = y^*$ and $y_1 \xleftarrow{R} \mathcal{Y}$, we get

$$\begin{aligned}
\mathsf{pPRFAdv}[\mathcal{B}, \mathcal{F}](\lambda, n) &= |\Pr[W_{\lambda,n,0}] - \Pr[W_{\lambda,n,1}]| \\
&= |\Pr[y' = y_0] - \Pr[y' = y_1]| \\
&= |\Pr[y' = y^*] - \Pr[y_1 = y' \mid y_1 \xleftarrow{R} [n]]| \\
&= \left|\Pr[\mathcal{A}(\mathsf{k_p}) = y^*] - \tfrac{1}{n}\right| \\
&= \mathsf{pPRFAdv}^{\mathrm{unp}}[\mathcal{A}, \mathcal{F}](\lambda, n)\,. \qquad \square
\end{aligned}$$

The standard PRF security requirement will also be useful. For an adversary $\mathcal{A}$ and a PRF $\mathcal{F}$, let $\mathsf{PRFGen}[\mathcal{A}, \mathcal{F}](\lambda, n)$ be the advantage of $\mathcal{A}$ in distinguishing an oracle $\mathcal{F}(\mathsf{k}, \cdot)$, for a key $k \leftarrow \mathsf{PRFGen}(1^\lambda, n)$, from an oracle $f(\cdot)$, for a random function $f\colon [n] \to [n]$. We say that a PRF is ϵ-secure if this advantage is at most $\epsilon(\lambda, n)$ for every efficient adversary.

Puncturable PRF from the tree-based PRF. We will use the puncturable PRF based on the classic tree-based PRF construction [GGM86] given by:

Theorem 32 ([BW13, KPTZ13, BGI14]). *Let* G *be an* ϵ_G*-secure length-doubling pseudorandom generator. Then, there exists an* $\epsilon_{\mathcal{F}}$*-secure puncturable PRF* $\mathcal{F}$ *with* $\epsilon_{\mathcal{F}}(\lambda, n) \leq O(\log n) \cdot \epsilon_G(\lambda)$*. Furthermore,* $\mathcal{F}$ *is an* $\epsilon'_{\mathcal{F}}$*-secure (standard) PRF with* $\epsilon'_{\mathcal{F}}(\lambda, n) \leq \mathrm{poly}(\lambda, \log n) \cdot \epsilon_G(\lambda)$*.*

A.3 Linearly homomorphic encryption

Our single-server offline/online PIR constructions (Section 5) rely on a linearly homomorphic encryption scheme [OU98, Pai99, DJ01]:

Definition 33 (Linearly homomorphic encryption over fields). Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption system with key space $\mathcal{K}$, message space $\mathbb{F}$ (for some finite field $\mathbb{F}$) and ciphertext space $\{\mathbb{G}_\lambda\}_{\lambda \in \mathbb{N}}$, where each $\mathbb{G}_\lambda$ is a finite group.

The scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *linearly homomorphic over field* $\mathbb{F}$ if, for all $\lambda \in \mathbb{N}$, all keys $k \leftarrow \mathsf{Gen}(1^\lambda)$, and all messages $m_0, m_1 \in \mathbb{F}$, we have

$$\mathsf{Dec}_k\big(\mathsf{Enc}_k(m_0) + \mathsf{Enc}_k(m_1)\big) = m_0 + m_1 \in \mathbb{F},$$

where the addition of ciphertexts is in the group $\mathbb{G}_\lambda$.

B Additional material on puncturable pseudorandom sets (Section 2)

B.1 Pseudorandomness of puncturable pseudorandom sets

The following proposition captures the property that puncturable pseudorandom sets contain s pseudorandom elements from $[n]$:

Proposition 34 (Puncturable pseudorandom set security imples pseudorandomness). *Let $\Psi = (\mathsf{Gen}, \mathsf{Punc}, \mathsf{Eval})$ be an ϵ-secure puncturable pseudorandom set with set size $s\colon \mathbb{N} \to \mathbb{N}$, and for every $\lambda, n \in \mathbb{N}$, define the two distributions:*

$$D_{\lambda,n,0} \coloneqq \big\{\mathsf{Eval}(\mathsf{Gen}(1^\lambda, n))\big\} \qquad \text{and} \qquad D_{\lambda,n,1} \coloneqq \left\{ S \xleftarrow{\text{R}} \binom{[n]}{s(n)} \right\}.$$

Then, for every efficient distinguishing adversary $\mathcal{A}$ and every $\lambda, n \in \mathbb{N}$,

$$\mathsf{DistAdv}[\mathcal{A}, D_{\lambda,n,0}, D_{\lambda,n,1}] \leq n \cdot s(n) \cdot \epsilon(\lambda, n)\,.$$

Proof. Let $\mathcal{A}$ be an algorithm that gets as input a set $S \in \binom{[n]}{s}$ and outputs a bit. (For brevity, we write s instead of $s(n)$.) We prove that $\mathcal{A}$'s distinguishing advantage must be negligible.

To do so, we define a sequence of $s+1$ hybrid distributions $\mathcal{H}_0, \ldots, \mathcal{H}_s$, each over the set $\binom{[n]}{s}$. These distributions interpolate between a truly random set and a set produced by the output of the puncturable pseudorandom set construction on a random key. In particular, in the ith hybrid distribution, we construct a set that consists of i elements from a pseudorandom set and $n - i$ random elements.

Formally, for $i = 0, 1, \ldots, s$, we define:

Hybrid $\mathcal{H}_i$.
- $S \leftarrow \mathsf{Eval}(\mathsf{Gen}(1^\lambda, n))$
- $S_0 \xleftarrow{\text{R}} \binom{S}{i}$
- $S_1 \xleftarrow{\text{R}} \binom{[n] \setminus S_0}{s - i}$
- Output $S' \leftarrow S_0 \cup S_1$.

We also define

$$\epsilon_i \coloneqq \Pr\left[\mathcal{A}(\mathcal{H}_i) = 1\right].$$

Note that $\mathcal{H}_0 \equiv D_{\lambda,n,1}$, and $\mathcal{H}_s \equiv D_{\lambda,n,0}$. Therefore, there exists an index $i \in [s]$ such that:

$$\epsilon_i - \epsilon_{i-1} \geq \frac{1}{s} \cdot \mathsf{DistAdv}[\mathcal{A}, D_{\lambda,n,0}, D_{\lambda,n,1}] \,. \tag{4}$$

Let $\mathcal{K}_\mathsf{p}$ be the key space of the puncturable pseudorandom set Ψ, and consider the following adversary $\mathcal{B}$ that attacks puncturable pseudorandom set Ψ as in the puncturable pseudorandom set security game (Game 1):

Adversary $\mathcal{B}$. Given a puncturable pseudorandom set key $\mathsf{sk}_\mathsf{p} \in \mathcal{K}_\mathsf{p}$ as input:

- $S_\mathsf{p} \leftarrow \mathsf{Eval}(\mathsf{sk}_\mathsf{p})$
- $S_0 \xleftarrow{\mathrm{R}} \binom{S_\mathsf{p}}{i-1}$
- $S_1 \xleftarrow{\mathrm{R}} \binom{[n] \setminus S_0}{s-i+1}$
- $S' \leftarrow S_0 \cup S_1$
- $b \leftarrow \mathcal{A}(S')$
- If $b = 1$: output $x' \xleftarrow{\mathrm{R}} S_1 \setminus S_\mathsf{p}$
- Else: output $x' \xleftarrow{\mathrm{R}} [n] \setminus S_\mathsf{p}$

Let $x^* \in [n]$ be the punctured element sampled by the challenger in Game 1. The guessing advantage of $\mathcal{B}$, as defined in Eq. (1), is then:

$$\begin{aligned} \mathsf{PSAdv}[\mathcal{B}, \Psi](\lambda, n(\lambda)) &= \Pr[x' = x^*] - \frac{1}{n-s+1} \\ &= \Pr[x' = x^* \wedge b = 0] + \Pr[x' = x^* \wedge b = 1] - \frac{1}{n-s+1} \,. \end{aligned} \tag{5}$$

We compute each of the first two terms separately. Observe that the set S', which $\mathcal{B}$ gives as input to $\mathcal{A}$, is distributed as in hybrid $\mathcal{H}_{i-1}$. Therefore,

$$\Pr[x' = x^* \wedge b = 0] = \Pr[x' = x^* \mid b = 0] \cdot \Pr[b = 0] = \frac{1}{n-s+1} \cdot (1 - \epsilon_{i-1}) \,. \tag{6}$$

For the second term, observe that, if $b = 1$ and $\mathcal{B}$ wins in the unpredictability game, the punctured element $x^* \in S_1$. Therefore:

$$\begin{aligned} &\Pr[x' = x^* \wedge b = 1] \\ =&\Pr[x' = x^* \wedge b = 1 \wedge x^* \in S_1] \\ =&\Pr[x' = x^* \mid b = 1 \wedge x^* \in S_1] \cdot \Pr[b = 1 \mid x^* \in S_1] \cdot \Pr[x^* \in S_1] \,. \end{aligned}$$

Let S be the unpunctured set sampled by the challenger in Game 1. Conditioned on $x^* \in S_1$, the set S', that $\mathcal{B}$ constructs, contains i elements chosen at random from S (namely the $i-1$ elements of S_0 and x^*) and $s-i$ random elements. Therefore, conditioned on $x^* \in S_1$, the set S' is distributed as in hybrid $\mathcal{H}_i$. Subsequently

$$\Pr[b = 1 \mid x^* \in S_1] = \epsilon_i \,,$$

and

$$\Pr[x' = x^* \wedge b = 1] = \epsilon_i \cdot \Pr\left[x' = x^* \middle| (b = 1) \wedge (x^* \in S_1)\right] \cdot \Pr[x^* \in S_1] \,.$$

We now use the law of total probability and condition on the size of the set $S_1 \setminus S_{\mathsf{p}}$. Since S_1 is chosen at random such that $|S_1 \setminus S_{\mathsf{p}}| = p$, the probability that $x^* \in S_1$ is $p/(n-s+1)$. Furthermore, when $x^* \in S_1$, and $\mathcal{B}$ chooses x' at random from $S_1 \setminus S_{\mathsf{p}}$, then $x' = x^*$ with probability $1/p$. Overall,

$$\begin{aligned}
&\Pr\left[x' = x^* \wedge b = 1\right] \\
&= \epsilon_i \cdot \sum_{p=1}^{s-i+1} \Big(\Pr\Big[x' = x^* \;\Big|\; (b=1) \wedge (x^* \in S_1) \wedge (|S_1 \setminus S_{\mathsf{p}}| = p)\Big] \\
&\qquad\qquad \cdot \Pr\Big[x^* \in S_1 \;\Big|\; |S_1 \setminus S_{\mathsf{p}}| = p\Big]\Big) \cdot \Pr\Big[\,|S_1 \setminus S_{\mathsf{p}}| = p\Big] \\
&= \epsilon_i \cdot \sum_{p=1}^{s-i+1} \frac{1}{p} \cdot \frac{p}{n-s+1} \cdot \Pr\Big[\,|S_1 \setminus S_{\mathsf{p}}| = p\Big] \\
&= \frac{\epsilon_i}{n-s+1}\,. \qquad (7)
\end{aligned}$$

Plugging (6) and (7) into (5), and using (4), we obtain that

$$\mathsf{PSAdv}[\mathcal{B}, \Psi](\lambda, n) = \frac{\epsilon_i - \epsilon_{i-1}}{n-s+1} \geq \frac{1}{s \cdot (n-s+1)} \cdot \mathsf{DistAdv}[\mathcal{A}, D_{\lambda,n,0}, D_{\lambda,n,1}](\lambda)\,.$$

□

B.2 Proof of Theorem 3

Let $\Psi_{\mathcal{F}}$ be the puncturable pseudorandom set construction instantiated with puncturable PRF $\mathcal{F}$. Recall that in Construction 4, we have set $s(n) \coloneqq \sqrt{n/2}$.

Efficiency. The routine $\Psi_{\mathcal{F}}.\mathsf{Gen}$ runs its inner loop at most λ times. At each iteration, the routine samples a puncturable PRF key and evaluates the puncturable PRF $s(n)$ times. Since the puncturable PRF is efficient, this inner loop runs in time $s(n) \cdot \mathrm{poly}(\lambda, \log n)$, so the entire $\Psi_{\mathcal{F}}.\mathsf{Gen}$ routine does as well.

The routine $\Psi_{\mathcal{F}}.\mathsf{Punc}$ evaluates the puncturable PRF at $s(n)$ points and then runs the PRF puncturing routine. Together, these operations require at most $s(n) \cdot \mathrm{poly}(\lambda, \log n)$ time.

The routine $\Psi_{\mathcal{F}}.\mathsf{Eval}$ evaluates the puncturable PRF $\mathcal{F}$ at $s(n)$ points. Each such evaluation takes at most $\mathrm{poly}(\lambda, \log n)$ time, for a total of $s(n) \cdot \mathrm{poly}(\lambda, \log n)$.

Correctness. The correctness of the puncturing routine follows by construction. As mentioned in Remark 5, to achieve perfect correctness, we slightly modify the construction to handle failures of the Gen routine. To this end, on inputs $\mathsf{sk} = \perp$ and $i \in [n]$, routine Punc outputs $\mathsf{sk}_{\mathsf{p}} \leftarrow (\perp, i)$. Similarly, on input $\mathsf{sk} = \perp$, routine Eval outputs the fixed set $[s(n)]$, and on input $\mathsf{sk}_{\mathsf{p}} = (\perp, i)$, routine Eval outputs the fixed set $[s(n)] \setminus \{i\}$. This guarantees perfect correctness.

Security. For brevity, let $s := s(n)$. We prove that for any efficient puncturable pseudorandom set adversary $\mathcal{A}$, there exist an efficient puncturable PRF adversary $\mathcal{B}$ and an efficient PRF adversary $\mathcal{B}_Z$, such that:

$$\begin{aligned}
\mathsf{PSAdv}[\mathcal{A}, \Psi_{\mathcal{F}}](\lambda, n) \leq\; & 4 \cdot \mathsf{pPRFAdv}^{\mathrm{unp}}[\mathcal{B}, \mathcal{F}](\lambda, n) \qquad (8) \\
& + O(\lambda) \cdot \mathsf{PRFAdv}[\mathcal{B}_Z, \mathcal{F}](\lambda, n, s) + 2^{-\lambda}\,.
\end{aligned}$$

Security will then follow from Lemma 31.

Consider now an adversary $\mathcal{B}$ that attacks the puncturable PRF $\mathcal{F}$ as in Game 30. For $\lambda, n \in \mathbb{N}$, adversary $\mathcal{B}$ operates as follows:

- Send a random value $x^* \xleftarrow{R} [s]$ to the challenger, where s is the set size for the punctured set.
- Receive the punctured PRF key $\mathsf{k_p}$ from the challenger.
- Compute the set

$$S^* = \{\mathsf{PRFEval}(\mathsf{k_p}, x) \mid x \in [s], x \neq x^*\}.$$

- If $|S^*| < s - 1$, output a random $y' \xleftarrow{R} [n]$.
- Otherwise, sample a bit $b \xleftarrow{R} \mathsf{Bernoulli}((s-1)/n)$:
  - If $b = 1$, output a random $y' \xleftarrow{R} S^*$.
  - If $b = 0$, run $\mathcal{A}$, using $(n, \mathsf{k_p})$ as the punctured set key. When $\mathcal{A}$ outputs a value $y' \in [n]$, outputs the same value.

Notation. Let k be the key of $\mathcal{F}$ chosen by the challenger in Game 30. We use the following notation in the proof:

- $y^* \leftarrow \mathsf{PRFEval}(\mathsf{k}, x^*)$ is the true value of $\mathcal{F}$ on the punctured point.
- $S^* \leftarrow \{\mathsf{PRFEval}(\mathsf{k}, x) : x \in [s], x \neq x^*\}$.
- We say that a key k is *good* if $|\{\mathsf{PRFEval}(\mathsf{k}, x) : x \in [s]\}| = s$.

Then, define the following three, mutually exclusive probability events:

- $\mathcal{E}_1$ is the event that $|S^*| < (s-1)$,
- $\mathcal{E}_2$ is the event that $|S^*| = (s-1)$ and $y^* \in S^*$, and
- $\mathcal{E}_3$ is the event that the key k is good. (Notice that $\mathcal{E}_3 \Leftrightarrow \neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2$.)

The key is often good. First, we argue that key k is often good (i.e., that event $\mathcal{E}_3$ happens often). In particular, let p_{good} be the probability that $|\{\mathsf{PRFEval}(\mathsf{k}, x) : x \in [s]\}| = s$. We argue there exists an efficient PRF adversary $\mathcal{B}_\perp$ such that

$$p_{\text{good}} \geq \frac{1}{2} - \mathsf{PRFAdv}[\mathcal{B}_\perp, \mathcal{F}](\lambda, n).$$

When $\mathcal{F}$ is a truly random function, a key is good when s *truly random* items sampled *with* replacement from $[n]$ are all distinct. The probability that two independent samples from $[n]$ are equal is $1/n$. The probability that there is a duplicate amongst s samples from $[n]$ is at most s^2/n (by the Union Bound). For $s = \sqrt{n/2}$ this probability is at most $1/2$. Since $\mathcal{F}$ is indistinguishable from a random function, there exists a PRF adversary $\mathcal{B}_\perp$ such that

$$|p_{\text{good}} - 1/2| \leq \mathsf{PRFAdv}[\mathcal{B}_\perp, \mathcal{F}](\lambda, n). \tag{9}$$

The main claim. We now argue that when:

- $\mathcal{E}_1$ or $\mathcal{E}_2$ occurs, $\mathcal{B}$'s guess is no worse than random and
- when $\mathcal{E}_3$ occurs, $\mathcal{B}$'s guess is significantly better than random.

Claim. *In particular, we claim that:*

$$\begin{aligned} &\text{(a)} && \Pr[y' = y^* \mid \mathcal{E}_1] = 1/n\,, \\ &\text{(b)} && \Pr[y' = y^* \mid \mathcal{E}_2] \geq 1/n\,, \end{aligned}$$

and that there exists an efficient puncturable pseudorandom set adversary $\mathcal{A}$ such that $\epsilon = \mathsf{PSAdv}[\mathcal{A}, \Psi_{\mathcal{F}}](\lambda, n)$ and

$$\text{(c)} \qquad \Pr[y' = y^* \mid \mathcal{E}_3] \geq \frac{1}{n} + \frac{\epsilon - (1 - p_{good})^\lambda}{2}.$$

Using the claim to prove security. We first show that the claim implies that security holds. Then we prove the claim. Assuming the claim for now, by the law of total probability, we have that:

$$\Pr[y' = y^*] = \sum_{t=1}^{3} \Pr[y' = y^* \mid \mathcal{E}_t] \cdot \Pr[\mathcal{E}_t]$$

and using parts (a), (b), and (c) of the claim

$$\begin{aligned} &\geq \left(\frac{1}{n}\right) \cdot \Pr[\mathcal{E}_1] + \left(\frac{1}{n}\right) \cdot \Pr[\mathcal{E}_2] + \left(\frac{1}{n} + \frac{\epsilon - (1 - p_{\text{good}})^\lambda}{2}\right) \cdot \Pr[\mathcal{E}_3], \\ &= \left(\frac{1}{n}\right) \cdot (\Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] + \Pr[\mathcal{E}_3]) + \left(\frac{\epsilon - (1 - p_{\text{good}})^\lambda}{2}\right) \cdot \Pr[\mathcal{E}_3] \\ &= \frac{1}{n} + \left(\frac{\epsilon - (1 - p_{\text{good}})^\lambda}{2}\right) \cdot \Pr[\mathcal{E}_3]. \end{aligned}$$

Now, let $\mathcal{B}_\perp$ be an algorithm as in Eq. (9). Then let $\epsilon' = \mathsf{PRFAdv}[\mathcal{B}_\perp, \mathcal{F}](\lambda, n)$ and and use the fact that $\Pr[\mathcal{E}_3] = p_{\text{good}} \geq 1/2 - \epsilon'$, as in to find:

$$\Pr[y = y^*] \geq \frac{1}{n} + \frac{\epsilon - (1/2 + \epsilon')^\lambda}{2} \cdot \left(\frac{1}{2} - \epsilon'\right).$$

Using the fact that $(1/2 + \epsilon')^\lambda \leq 1/2 + \lambda\epsilon'$, we have that

$$\begin{aligned} \mathsf{pPRFAdv}^{\text{unp}}[\mathcal{B}, \mathcal{F}](\lambda, n) &= \Pr[y = y^*] - \frac{1}{n} \\ &\geq \left[\frac{1}{n} + \frac{\epsilon - 2^{-\lambda} - \lambda\epsilon'}{2} \cdot \left(\frac{1}{2} - \epsilon'\right)\right] - \frac{1}{n} \\ &\geq \frac{\epsilon - 2^{-\lambda} - \lambda\epsilon'}{2} \cdot \left(\frac{1}{2} - \epsilon'\right) \\ &\geq \frac{\epsilon}{4} - 2^{-\lambda - 2} - \frac{(\lambda + 1)\epsilon'}{2}. \end{aligned}$$

By rearranging, we complete the proof of security:

$$\epsilon \leq 4 \cdot \mathsf{pPRFAdv}^{\text{unp}}[\mathcal{B}, \mathcal{F}](\lambda, n) + 2(\lambda + 1) \cdot \mathsf{PRFAdv}[\mathcal{B}_\perp, \mathcal{F}](\lambda, n) + 2^{-\lambda}.$$

Proving the claim. Now our task is to prove parts (a)–(c) of the claim.

(a) When $\mathcal{E}_1$ occurs, adversary $\mathcal{B}$ outputs a random guess, and thus

$$\Pr[y' = y^* \mid \mathcal{E}_1] = \frac{1}{n}\,.$$

(b) When $\mathcal{E}_2$ occurs, the probability of $y' = y^*$ is at least that of adversary $\mathcal{B}$ sampling $b = 1$ and correctly guessing y^* from S^*. Therefore,

$$\Pr[y' = y^* \mid \mathcal{E}_2] \geq \frac{s-1}{n} \cdot \frac{1}{s-1} = \frac{1}{n}\,.$$

(c) Finally, when $\mathcal{E}_3$ occurs, the probability of $y' = y^*$ is at least that of adversary $\mathcal{B}$ sampling $b = 0$ and then getting the right answer from adversary $\mathcal{A}$. Therefore,

$$\Pr[y' = y^* \mid \mathcal{E}_3] \geq \left(1 - \frac{s-1}{n}\right) \cdot \Pr\left[\mathcal{A}(n, \mathsf{k_p}) = y^* \;\middle|\; \begin{array}{c} \mathsf{k} \leftarrow \mathsf{PRFGen}(1^\lambda, n) \\ \mathsf{k} \text{ is good} \\ x^* \xleftarrow{R} [s] \\ \mathsf{k_p} \leftarrow \mathsf{PRFPunc}(\mathsf{k}, x^*) \\ y^* \leftarrow \mathsf{PRFEval}(\mathsf{k}, x^*) \end{array}\right].$$

Let W be the event that $\mathcal{A}$ wins in the puncturable pseudorandom set security game (Game 1). Conditioned on the key k being good, the distributions (n, k) for $\mathsf{k} \leftarrow \mathsf{PRFGen}(1^\lambda, n)$ and (n, k) for $\mathsf{k} \leftarrow \Psi_{\mathcal{F}}.\mathsf{Gen}(1^\lambda, n)$ are identical. Therefore,

$$\Pr[y' = y^* \mid \mathcal{E}_3] \geq \left(1 - \frac{s-1}{n}\right) \cdot \Pr[W \mid \mathsf{k} \text{ is good}]. \tag{10}$$

Here, we are conditioning on the event that a key k sampled from $\Psi_{\mathcal{F}}.\mathsf{Gen}(1^\lambda, n)$ is good.
We know that

$$\Pr[W] = \Pr[W \mid \mathsf{k} \text{ is good}] \cdot \Pr[\mathsf{k} \text{ is good}] + \Pr[W \mid \mathsf{k} \text{ is not good}] \cdot \Pr[\mathsf{k} \text{ is not good}],$$

or

$$\begin{aligned} \Pr[W \mid \mathsf{k} \text{ is good}] &\geq \Pr[W] - \Pr[W \mid \mathsf{k} \text{ is not good}] \cdot \Pr[\mathsf{k} \text{ is not good}], \\ &\geq \Pr[W] - \Pr[\mathsf{k} \text{ is not good}]. \end{aligned} \tag{11}$$

By the definition of the puncturable pseudorandom set security advantage, we have

$$\Pr[W] = \frac{1}{n-s+1} + \epsilon, \tag{12}$$

and with Eqs. (10)-(12), we have:

$$\Pr[y' = y^* \mid \mathcal{E}_3] \geq \left(1 - \frac{s-1}{n}\right) \cdot \left(\frac{1}{n-s+1} + \epsilon - \Pr[\mathsf{k} \text{ is not good}])\right).$$

Using the fact that $s = \sqrt{n/2}$ and thus $n - s + 1 \geq n/2$, we have

$$\Pr[y' = y^* \mid \mathcal{E}_3] \geq \cdot \left(\frac{1}{n} + \epsilon/2 - \frac{1}{2} \cdot \Pr[\mathsf{k} \text{ is not good}] \right).$$

By construction, the probability that a key k generated by $\Psi_{\mathcal{F}}.\mathsf{Gen}$ is not good, by Eq. (9), and the fact that $\Psi_{\mathcal{F}}.\mathsf{Gen}$ samples λ independent random puncturable PRF keys before failing, is at most $(1 - p_{\text{good}})^{\lambda}$. Then

$$\Pr[y' = y^* \mid \mathcal{E}_3] \geq \frac{1}{n} + \frac{\epsilon - (1 - p_{\text{good}})^{\lambda}}{2}.$$

This completes the proof of the claim.

B.3 Proof of Corollary 6

Let G be an ϵ_G-secure length-doubling PRG. By Theorem 32, there exists a puncturable PRF $\mathcal{F}_G$, that is $\epsilon_{\mathcal{F}_G}$-secure for

$$\epsilon_{\mathcal{F}_G}(\lambda, n) \leq O(\log n) \cdot \epsilon_G(\lambda).$$

Furthermore, $\mathcal{F}_G$ is an $\epsilon'_{\mathcal{F}_G}$-secure (standard) PRF for

$$\epsilon'_{\mathcal{F}_G}(\lambda, n) \leq \text{poly}(\lambda, \log n) \cdot \epsilon_G(\lambda).$$

By Theorem 3, there exists a puncturable pseudorandom set scheme $\Psi_{\mathcal{F}_G}$ that is ϵ-secure for

$$\begin{aligned} \epsilon(\lambda, n) &\leq 4\epsilon_{\mathcal{F}_G}(\lambda, n) + O(\lambda) \cdot \epsilon'_{\mathcal{F}_G}(\lambda, n) + 2^{-\lambda} \\ &\leq \text{poly}(\lambda, n) \cdot (\epsilon_G(\lambda) + 2^{-\lambda}). \end{aligned}$$

B.4 Proof sketch of Theorem 7

To obtain a puncturable pseudorandom set with a fast membership test, define a puncturable pseudorandom set whose elements are the evaluations of a pseudorandom permutation $P(\mathsf{sk}, \cdot)$ on the points $\{1, \ldots, s\}$. That is, a set key is a PRP key sk, and $\Psi.\mathsf{Eval}(\mathsf{sk}) = \{P(\mathsf{sk}, i) \mid i \in [s]\}$. A punctured set key, on the other hand, is a fully explicit representation of the punctured set itself, as a list of $s - 1$ elements, Since a PRP is efficiently invertible given the key, it is easy to test, given i and sk, whether $i \in \Psi.\mathsf{Eval}(\mathsf{sk})$. To do so, compute $y \leftarrow P^{-1}(\mathsf{sk}, i)$ and test whether $y \in [s]$.

We construct the puncturable pseudorandom set as follows:

- $\Psi_P.\mathsf{Gen}(1^{\lambda}, n) \to \mathsf{sk}$. Sample a key sk for a pseudorandom permutation $P(\mathsf{sk}, \cdot)$ on security parameter λ that maps $[n]$ to $[n]$.
- $\Psi_P.\mathsf{Punc}(\mathsf{sk}, i) \to \mathsf{sk}_\mathsf{p}$. Output $\mathsf{sk}_\mathsf{p} \leftarrow \Psi_P.\mathsf{Eval}(\mathsf{sk}) \setminus \{i\}$ as a list of $s-1$ integers of $\log n$ bits each.
- $\Psi_P.\mathsf{Eval}(\mathsf{sk}) \to S$. Output $\{P(\mathsf{sk}, i) \mid i \in [s]\}$.

– $\Psi_P.\mathsf{InSet}(\mathsf{sk}, i)$. Compute $y \leftarrow P^{-1}(\mathsf{sk}, i)$. If $y \in [s]$, output "1." Otherwise, output "0."

The correctness and efficiency properties follow immediately. Security follows from PRP security: an adversary that can win the puncturable pseudorandom set security game can guess the value of a PRP on an unqueried point and thus break PRP security.

B.5 Shifting puncturable pseudorandom sets

We now show how to equip any puncturable set with the additional routines $\mathsf{GenWith}$ and Shift, which we introduced in Section 2.3.

Given a puncturable pseudorandom set $\Psi = (\mathsf{Gen}, \mathsf{Punc}, \mathsf{Eval})$, defined over key spaces $\mathcal{K}$ and $\mathcal{K}_\mathsf{p}$, define a new puncturable pseudorandom set $\Psi' = (\mathsf{Gen}', \mathsf{Punc}', \mathsf{Eval}')$ extended by routines $(\mathsf{GenWith}, \mathsf{Shift})$, with key spaces $\mathcal{K} \times \mathbb{N}$ and $\mathcal{K}_\mathsf{p} \times \mathbb{N}$, where

– $\mathsf{Gen}'(1^\lambda, n)$ samples $\Delta \xleftarrow{R} [n]$ and outputs $\mathsf{sk} \leftarrow (\Psi.\mathsf{Gen}(1^\lambda, n), \Delta)$,
– $\mathsf{Punc}'((\mathsf{sk}, \Delta), i)$ outputs $\mathsf{sk}_\mathsf{p} \leftarrow (\mathsf{Punc}(\mathsf{sk}, i - \Delta), \Delta)$,
– $\mathsf{Eval}'((\mathsf{sk}, \Delta))$ outputs $S \leftarrow \{i + \Delta : i \in \mathsf{Eval}(\mathsf{sk})\}$,
– $\mathsf{Shift}((\mathsf{sk}, \Delta), j)$ outputs $\mathsf{sk}' \leftarrow (\mathsf{sk}, \Delta + j \bmod n)$, and
– $\mathsf{GenWith}(1^\lambda, n, i)$ runs $\mathsf{sk} \leftarrow \mathsf{Gen}'(1^\lambda, n)$, chooses $i' \xleftarrow{R} \mathsf{Eval}(\mathsf{sk})$ and outputs $\mathsf{Shift}(\mathsf{sk}, i - i')$.

By inspection, the extended construction satisfies the correction requirement. Moreover, the following holds:

Proposition 35. *Let Ψ be a puncturable pseudorandom set and Ψ' be the extension above. Then*

1. *Ψ' is a secure puncturable pseudorandom set.*
2. *For every $\lambda, n \in \mathbb{N}$, the following two distributions are identical:*

$$\left\{\begin{array}{l} i \xleftarrow{R} [n] \\ \mathsf{sk} \leftarrow \mathsf{GenWith}(1^\lambda, n, i) \\ \text{output } (i, \mathsf{sk}) \end{array}\right\} \equiv \left\{\begin{array}{l} \mathsf{sk} \leftarrow \mathsf{Gen}'(1^\lambda, n) \\ S \leftarrow \mathsf{Eval}'(\mathsf{sk}) \\ i \xleftarrow{R} S \\ \text{output } (i, \mathsf{sk}) \end{array}\right\}.$$

Proof. For the first part, consider an adversary $\mathcal{A}$ that attacks Ψ' in Experiment 1. Now let $\mathcal{B}$ be an adversary that attacks Ψ as follows. On input $\mathsf{sk}_\mathsf{p} \in \mathcal{K}_\mathsf{p}$ from its challenger, adversary $\mathcal{B}$ chooses $\Delta \xleftarrow{R} [n]$, sets $\mathsf{sk}'_\mathsf{p} \leftarrow (\mathsf{sk}_\mathsf{p}, \Delta)$ and runs $\mathcal{A}$ on sk'_p. When $\mathcal{A}$ outputs its guess $j \in [n]$, adversary $\mathcal{B}$ outputs $i \leftarrow j - \Delta$. Adversary $\mathcal{B}$ wins with exactly the same probability as $\mathcal{A}$, which proves the first part of the proposition.

For the second part,

$$\left\{\begin{array}{l} i \xleftarrow{R} [n] \\ \mathsf{sk} \leftarrow \mathsf{GenWith}(1^\lambda, n, i) \\ \text{output } (i, \mathsf{sk}) \end{array}\right\} \equiv \left\{\begin{array}{l} i \xleftarrow{R} [n] \\ \mathsf{sk}' \leftarrow \mathsf{Gen}'(1^\lambda, n) \\ i' \xleftarrow{R} \mathsf{Eval}'(\mathsf{sk}') \\ \mathsf{sk} \leftarrow \mathsf{Shift}(\mathsf{sk}', i - i') \\ \text{output } (i, \mathsf{sk}) \end{array}\right\} \quad \left(\begin{array}{l}\text{by definition} \\ \text{of } \mathsf{GenWith}\end{array}\right)$$

$$
\equiv \left\{ \begin{array}{l} i \xleftarrow{R} [n] \\ \mathsf{sk}_0 \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \Delta' \xleftarrow{R} [n] \\ i' \xleftarrow{R} \mathsf{Eval}'((\mathsf{sk}_0, \Delta')) \\ \mathsf{sk} \leftarrow \mathsf{Shift}((\mathsf{sk}_0, \Delta'), i - i') \\ \text{output } (i, \mathsf{sk}) \end{array} \right\} \quad \left(\begin{array}{l} \text{by definition} \\ \text{of } \mathsf{Gen}' \end{array} \right)
$$

$$
\equiv \left\{ \begin{array}{l} i \xleftarrow{R} [n] \\ \mathsf{sk}_0 \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \Delta' \xleftarrow{R} [n] \\ i'_0 \xleftarrow{R} \mathsf{Eval}(\mathsf{sk}_0) \\ i' \leftarrow i'_0 + \Delta' \\ \mathsf{sk} \leftarrow \mathsf{Shift}((\mathsf{sk}_0, \Delta'), i - i') \\ \text{output } (i, \mathsf{sk}) \end{array} \right\} \quad \left(\begin{array}{l} \text{by definition} \\ \text{of } \mathsf{Eval}' \end{array} \right)
$$

$$
\equiv \left\{ \begin{array}{l} i \xleftarrow{R} [n] \\ \mathsf{sk}_0 \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \Delta' \xleftarrow{R} [n] \\ i'_0 \xleftarrow{R} \mathsf{Eval}(\mathsf{sk}_0) \\ i' \leftarrow i'_0 + \Delta' \\ \Delta \leftarrow \Delta' + i - i' \\ \text{output } (i, (\mathsf{sk}_0, \Delta)) \end{array} \right\} \quad \left(\begin{array}{l} \text{by definition} \\ \text{of } \mathsf{Shift} \end{array} \right)
$$

$$
\equiv \left\{ \begin{array}{l} i \xleftarrow{R} [n] \\ \mathsf{sk}_0 \leftarrow \mathsf{Gen}(1^\lambda, n) \\ i'_0 \xleftarrow{R} \mathsf{Eval}(\mathsf{sk}_0) \\ \Delta \leftarrow i - i'_0 \\ \text{output } (i, (\mathsf{sk}_0, \Delta)) \end{array} \right\} \quad \left(\begin{array}{l} \text{plugging } i' \text{ into } \Delta \\ \text{cancels out } \Delta' \end{array} \right)
$$

$$
\equiv \left\{ \begin{array}{l} \mathsf{sk}_0 \leftarrow \mathsf{Gen}(1^\lambda, n) \\ i'_0 \xleftarrow{R} \mathsf{Eval}(\mathsf{sk}_0) \\ \Delta \xleftarrow{R} [n] \\ i \xleftarrow{R} \Delta + i'_0 \\ \text{output } (i, (\mathsf{sk}_0, \Delta) \end{array} \right\} \quad \left(\begin{array}{l} \text{by swapping the} \\ \text{order of sampling} \\ \text{of } \Delta \text{ and } i \end{array} \right)
$$

$$
\equiv \left\{ \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{Gen}'(1^\lambda, n) \\ i \xleftarrow{R} \mathsf{Eval}'(\mathsf{sk}) \\ \text{output } (i, \mathsf{sk}) \end{array} \right\} \quad \text{(by defintion of } \mathsf{Gen}'\text{).} \qquad \square
$$

B.6 A key lemma

The next lemma will be used to prove the security of our PIR constructions. At a very high level, the lemma shows that it is possible to sample a set key sk and punctured set key sk_p in such a way that

- a chosen element $i \in [n]$ is in the set $\mathsf{Eval}(\mathsf{sk})$, and
- the punctured set key sk_p completely hides the chosen point i.

Lemma 36. *Let $\Psi = (\mathsf{Gen}, \mathsf{Punc}, \mathsf{Eval})$ be a puncturable pseudorandom set extended by $(\mathsf{Shift}, \mathsf{GenWith})$, with set size $s(n)$. For every $\lambda, n \in \mathbb{N}$, and every*

$i \in [n]$, *define the distribution*

$$D_{\lambda,n,i} = \left\{ \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{GenWith}(1^\lambda, n, i) \\ S \leftarrow \mathsf{Eval}(\mathsf{sk}) \\ b \xleftarrow{R} \mathsf{Bernoulli}((s(n)-1)/n) \\ \text{if } b = 0 : \quad i_{\mathsf{punc}} \leftarrow i \\ \text{if } b = 1 : \quad i_{\mathsf{punc}} \xleftarrow{R} S \setminus \{i\} \\ \text{output } \mathsf{sk}_{\mathsf{p}} \leftarrow \mathsf{Punc}(\mathsf{sk}, i_{\mathsf{punc}}) \end{array} \right\} .$$

Then, for every polynomially bounded $n = n(\lambda)$ and every $i, j \in [n]$, it holds that

$$\{D_{\lambda,n,i}\}_{\lambda\in\mathbb{N}} \approx_c \{D_{\lambda,n,j}\}_{\lambda\in\mathbb{N}} .$$

Specifically, for every efficient distinguishing adversary $\mathcal{A}$, there exists a (non-uniform) efficient puncturable pseudorandom set adversary $\mathcal{B}$, such that for every $\lambda, n \in \mathbb{N}$

$$\max_{i,j\in[n]} \mathsf{DistAdv}[\mathcal{A}, D_{\lambda,n,i}, D_{\lambda,n,j}] \leq n^2 \cdot \mathsf{PSAdv}[\mathcal{B}, \Psi](\lambda, n) .$$

Proof. Let $\mathcal{K}_{\mathsf{p}}$ be the punctured-key space of Ψ, and let $\lambda, n \in \mathbb{N}$. For brevity, we use s to denote $s(n)$, and D_i to denote $D_{i,\lambda,n}$ for any $i \in [n]$.

Consider a distinguishing adversary $\mathcal{A}$ as in the statement of the lemma. By Proposition 25, there exists a non-uniform guessing adversary $\mathcal{A}'$ that takes as input a punctured set key $\mathsf{sk}_{\mathsf{p}} \in \mathcal{K}_{\mathsf{p}}$ and outputs an integer from 1 to n such that

$$\Pr\left[\mathcal{A}'(\mathsf{sk}_{\mathsf{p}}) = i \,:\, i \xleftarrow{R} [n], \mathsf{sk}_{\mathsf{p}} \xleftarrow{R} D_i\right] \geq \frac{1}{n} + \frac{2}{n^2} \cdot \max_{i,j\in[n]} \mathsf{DistAdv}[\mathcal{A}, D_i, D_j] . \quad (13)$$

Moreover, the running time of $\mathcal{A}'$ is linear in the running time of $\mathcal{A}$.

Consider the following three experiments. In each experiment, we generate a punctured key $\mathsf{sk}_{\mathsf{p}} \in \mathcal{K}_{\mathsf{p}}$ and give it to the adversary $\mathcal{A}'$. The adversary then outputs a value $i_{\mathsf{adv}} \in [n]$. The adversary $\mathcal{A}'$ wins if $i_{\mathsf{adv}} = i$ where we choose $i \in [n]$ in the experiment. (The exact way in which we choose i differs between the experiments.) For $\ell \in \{0, 1, 2\}$, let W_ℓ be the probability that $\mathcal{A}'$ wins in Experiment ℓ.

Experiment 0. The adversary's view in this experiment is distributed exactly as in distribution D_i. In particular, we choose $i \xleftarrow{R} [n]$, generates $\mathsf{sk}_{\mathsf{p}} \in \mathcal{K}_{\mathsf{p}}$ as in distribution D_i and run the adversary on sk_{p}, who outputs an index $i_{\mathsf{adv}} \in [n]$.

Experiment 1. We modify Experiment 0 as follows:

$\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)$
$S \leftarrow \mathsf{Eval}(\mathsf{sk})$
$i \xleftarrow{R} S$
sample a bit $b \xleftarrow{R} \mathsf{Bernoulli}((s-1)/n)$
if $b = 0$ then $i_{\mathsf{punc}} \leftarrow i$ else $i_{\mathsf{punc}} \xleftarrow{R} S \setminus \{i\}$
output $\mathsf{sk}_{\mathsf{p}} \leftarrow \mathsf{Punc}(\mathsf{sk}, i_{\mathsf{punc}})$

By the third part of Proposition 35, the pairs (i, sk) in Experiments 0 and 1 are identically distributed. Therefore $W_0 = W_1$.

Experiment 2. Note that i_{punc} is distributed uniformly over S and that our random choice of i_{punc} in the experiment is independent of the adversary's view. We can therefore defer the random choice of the index i to after having run the adversary. In this new experiment, we first sample $\mathsf{sk}_{\mathsf{p}} \in \mathcal{K}_{\mathsf{p}}$:

$\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)$
$S \leftarrow \mathsf{Eval}(\mathsf{sk})$
$i_{\mathsf{punc}} \xleftarrow{\mathrm{R}} S$
output $\mathsf{sk}_{\mathsf{p}} \leftarrow \mathsf{Punc}(\mathsf{sk}, i_{\mathsf{punc}})$

We then run the adversary on sk_{p}, who computes $i_{\mathsf{adv}} \leftarrow \mathcal{A}'(\mathsf{sk}_{\mathsf{p}})$. At this point, we execute the following steps in the experiment:

sample a bit $b \xleftarrow{\mathrm{R}} \mathsf{Bernoulli}((s-1)/n)$
if $b = 0$ then set $i \leftarrow i_{\mathsf{punc}}$
if $b = 1$ then set $i \xleftarrow{\mathrm{R}} S \setminus \{i_{\mathsf{punc}}\}$

Since the joint distribution of (i, i_{punc}) in Experiments 1 and 2 is identical, we have that

$$\Pr[W_1] = \Pr[W_2] = \Pr[i{=}i_{\mathsf{adv}}] = p_0 + p_1,$$

where

$$p_0 = \Pr[b{=}0] \cdot \Pr[i_{\mathsf{adv}}{=}i_{\mathsf{punc}}] = \tfrac{n-s+1}{n} \cdot \Pr[i_{\mathsf{adv}}{=}i_{\mathsf{punc}}]\,,$$

and

$$\begin{aligned} p_1 &= \Pr[b{=}1] \cdot \Pr[i_{\mathsf{adv}} \in (S \setminus \{i_{\mathsf{punc}}\})] \cdot \Pr[i{=}i_{\mathsf{adv}} \mid i_{\mathsf{adv}} \in (S \setminus \{i_{\mathsf{punc}}\})] \\ &= \frac{s-1}{n} \cdot \Pr[i_{\mathsf{adv}} \in \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})] \cdot \frac{1}{s-1} \\ &= \frac{1}{n} \cdot \Pr[i_{\mathsf{adv}} \in \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})]\,. \end{aligned}$$

In the last part, we used the fact that $\mathsf{Eval}(\mathsf{sk}_{\mathsf{p}}) = S \setminus \{i_{\mathsf{punc}}\}$. Overall, we obtain

$$\Pr[W_2] = \tfrac{n-s+1}{n} \cdot \Big(\Pr[i_{\mathsf{adv}}{=}i_{\mathsf{punc}}] + \tfrac{1}{n-s+1} \cdot \Pr[i_{\mathsf{adv}} \in \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})] \Big)\,. \tag{14}$$

We now construct the adversary $\mathcal{B}$ that attacks the underlying puncturable pseudorandom set in the puncturable pseudorandom set security experiment (Experiment 1). On input $\mathsf{sk}_{\mathsf{p}} \in \mathcal{K}_{\mathsf{p}}$, adversary $\mathcal{B}$ proceeds as follows:

$i_{\mathsf{adv}} \leftarrow \mathcal{A}'(\mathsf{sk}_{\mathsf{p}})$
if $i_{\mathsf{adv}} \notin \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})$ then output $i_{\mathcal{B}} \leftarrow i_{\mathsf{adv}}$
else output $i_{\mathcal{B}} \xleftarrow{\mathrm{R}} [n] \setminus \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})$

Note that since $\mathcal{A}'$ is a non-uniform algorithm, so is the constructed $\mathcal{B}$.

Recall that in Experiment 1, we say that $\mathcal{B}$ wins if $i_{\mathcal{B}} = i_{\mathsf{punc}}$, and let $W_{\mathcal{B}}$ be the probability of that. Let Z be the event that $i_{\mathsf{adv}} \in \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})$. Then

$$\Pr[W_{\mathcal{B}}] = \Pr[i_{\mathcal{B}} = i_{\mathsf{punc}} \wedge \neg Z] + \Pr[i_{\mathcal{B}} = i_{\mathsf{punc}} \mid Z] \cdot \Pr[Z].$$

Since $i_{\mathcal{B}}$ is random in $[n] \setminus \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})$ when $i_{\mathsf{adv}} \in \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})$, we have that

$$\Pr[i_{\mathcal{B}} = i_{\mathsf{punc}} \mid Z] = \frac{1}{n-s-1}.$$

Furthermore, since $i_{\mathsf{punc}} \notin \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})$, the probability event $i_{\mathcal{B}} = i_{\mathsf{punc}}$ implies the event $\neg Z$. Therefore,

$$\begin{aligned}
\Pr[W_{\mathcal{B}}] &= \Pr[i_{\mathsf{adv}} = i_{\mathsf{punc}}] + \frac{1}{n-s-1} \cdot \Pr[Z] \\
&= \Pr[i_{\mathsf{adv}} = i_{\mathsf{punc}}] + \Pr[i_{\mathsf{adv}} \in \mathsf{Eval}(\mathsf{sk}_{\mathsf{p}})] \cdot \tfrac{1}{n-s+1} \\
&= \Pr[W_2] \cdot \frac{n}{n-s+1} \qquad \text{(by Eq. (14))}\,.
\end{aligned}$$

Previously, we have shown that $W_0 = W_1 = W_2$. Therefore

$$\begin{aligned}
\mathsf{PSAdv}[\mathcal{B}, \Psi](\lambda, n) &= \Pr[W_{\mathcal{B}}] - \frac{1}{n-s+1} \\
&= \Pr[W_2] \cdot \frac{n}{n-s+1} - \frac{1}{n-s+1} \\
&\geq \Pr[W_0] \cdot \frac{n}{n-s+1} - \frac{1}{n-s+1}
\end{aligned}$$

From Eq. (13), we get:

$$\begin{aligned}
\mathsf{PSAdv}[\mathcal{B}, \Psi](\lambda, n) &\geq \left(\tfrac{1}{n} + \max_{i,j \in [n]} \mathsf{DistAdv}[\mathcal{A}, D_i, D_j] \cdot \tfrac{2}{n^2} \right) \cdot \tfrac{n}{n-s+1} - \tfrac{1}{n-s+1} \\
&\geq \max_{i,j \in [n]} \mathsf{DistAdv}[\mathcal{A}, D_i, D_j] \cdot \tfrac{2}{n^2}.
\end{aligned}$$

□

C Additional material on the two-server case (Section 3)

C.1 Discussion of two-server offline/online PIR

Remark 37 (Larger database rows). In practical applications, the database may consist of a vector of n ℓ-bit strings, for $\ell > 1$. In this case, the client and server can run the PIR protocol of Theorem 14 ℓ times in parallel to yield a PIR scheme over this larger database. If ℓ grows with n (e.g., $\ell = n^{1/4}$), more efficient techniques apply [CGKS95, Section 6].

Remark 38 (Offline/online PIR as a random self-reduction). We can view Theorem 11 as a kind of nonuniform random-self reduction [AFK89, FF93]. In particular, say that you are given oracle access to a function $f : \{0,1\}^n \to \{0,1\}$ and $\widetilde{O}(\sqrt{n})$ bits of advice about f. Is it possible to compute $f(x)$ using $\widetilde{O}(\sqrt{n})$ queries to f, such that the distribution of your queries is independent of x?

Theorem 11 implies that the answer to this question is "yes," and our lower bound (Theorem 23) shows that a better advice- and query-complexity trade-off is essentially impossible.

Remark 39 (Reducing client online time). In the online phase of our computational two-server PIR construction, the client runs in $\widetilde{O}(\sqrt{n})$ time. In particular, the client has a fixed sets $S, \Delta \subseteq [n]$ of size $\widetilde{\Theta}(\sqrt{n})$ (chosen in the offline phase) and and an index $i_{\mathsf{pir}} \in [n]$ (chosen in the online phase) and the client must find values $s \in S$ and $\delta \in \Delta$ such that $i_{\mathsf{pir}} = s + \delta \bmod n$.

This is almost precisely the *indexing 3-SUM problem* [GKLP17]. The simple algorithm we sketched uses client time and storage $\widetilde{O}_\lambda(n^{1/2})$. Golovnev et al. [GGH+19] point out that it is possible to solve 3-SUM indexing faster with preprocessing using Fiat and Naor's refinement [FN00] of Hellman tables [Hel80].

Applying this result to our setting: if the client performs $\widetilde{O}_\lambda(n)$ work in the offline phase, and stores a data structure of size $S > 0$, it can complete the online phase in time $T = \widetilde{O}_\lambda(n^2/S^2)$. (The result of Golovnev et al. [GGH+19] actually gives a slightly worse trade-off of $T = \widetilde{O}(n^3/S^3)$. Since our 3SUM-indexing instances are chosen at random, we can get a slightly better trade-off by applying the result of Fiat and Naor that applies to random functions.)

So, for example, if the client stores a data structure of size $S = \widetilde{O}_\lambda(n^{7/8})$, it can process online requests in time $T = \widetilde{O}_\lambda(n^{1/4})$, after doing $\widetilde{\Omega}_\lambda(n)$ offline work. It would be excellent to find a way to construct this data structure in sublinear time $o(n)$, since otherwise the linear-time client-side preprocessing cost is burdensome.

C.2 Proof of Lemma 15

We prove that Construction 16 satisfies the requirements of Definition 8 with the efficiency parameters given in Lemma 15.

Claim (Correctness). *For every $\lambda, n \in \mathbb{N}$, every database $x \in \{0,1\}^n$, and every $i \in [n]$, the client succeeds in retrieving the ith bit of the database with probability 1.*

Proof. We first analyze the failure probability of the PIR scheme as if the client were using a perfectly secure puncturable pseudorandom set. Reading index $i \in [n]$ fails when $j = \perp$ or when $i_{\mathsf{punc}} \neq i$. The probability of the first failure event is:

$$\begin{aligned}
\Pr[j = \perp] &= \Pr\left[\wedge_{j=1}^{m} (i - \delta_j \notin \mathsf{Eval}(\mathsf{sk}))\right] \\
&= \Pr\left[\delta_j \notin \mathsf{Eval}(\mathsf{sk})\right]^m \quad \text{(since } \delta_1, \ldots, \delta_m \text{ are random)} \\
&= \left(1 - \frac{s}{n}\right)^m = \left(1 - \frac{s}{n}\right)^{\frac{n}{s}\log n} \leq e^{-\log n} = \frac{1}{n}. \qquad (15)
\end{aligned}$$

The second failure event, $i_{\mathsf{punc}} \neq i$, occurs when the client's random bit $b = 1$ (in the Query algorithm), the probability of which is $(s-1)/n$. Therefore, each read fails with probability at most $(s-1)/n + 1/n \leq 1/2$, where we have used the assumption that $s \leq n/2$.

Since the client can detect whenever a failure occurs, by running λ instances of the scheme in parallel, using independent randomness for each instance, we

can drive the overall failure probability (when the puncturable pseudorandom set is perfectly secure) to $2^{-\lambda}$. To avoid leaking partial information to the server, the number of repetitions needs to be fixed and oblivious of any early success.

By Proposition 34, each of the λ puncturable pseudorandom sets $\mathsf{Eval}(\mathsf{sk})$ (one at each instance) is computationally indistinguishable from a perfectly secure puncturable pseudorandom set. Therefore, the failure probability of all λ instances must be negligibly close to $2^{-\lambda}$.

To achieve perfect correctness, if the client detects an error (which happens with only a negligible probability), it simply reads its desired bit from the database using a non-private lookup. Now, the client *always* receives its desired bit, at the cost of a negligible loss in security. □

Claim (Server efficiency). *The server's offline-phase running time is* $n \cdot \mathrm{poly}(\lambda, \log n)$ *and the server's online-phase running time is* $s(n) \cdot \mathrm{poly}(\lambda, \log n)$.

Proof. The offline server's work consists of computing $m = (n/s(n)) \log n$ sums of the form $\sum_{i \in S} x_{i+j}$ where $S \leftarrow \mathsf{Eval}(\mathsf{sk})$ is a set of size $s(n)$. Computing the set S by evaluating the puncturable pseudorandom set takes time $s(n) \cdot \mathrm{poly}(\lambda, \log n)$, and given S, each sum can be computed in time $O(s(n) \log n)$. This yields a total offline server running time of $n \cdot \mathrm{poly}(\lambda, \log n)$.

The online server computes only one such sum, and therefore its running time is $s(n) \cdot \mathrm{poly}(\lambda, \log n)$. □

Claim (Client efficiency). *The client runs in time* $\widetilde{O}(s(n) + n/s(n)) \cdot \mathrm{poly}(\lambda)$ *and stores* $\lambda \cdot |\kappa| + (\lambda n/s(n)) \log^2 n$ *bits between the offline and online phases.*

Proof. The client runs three routines:

- In the offline phase, the client generates a puncturable pseudorandom set key for a set of size $s(n)$. By the efficiency property of a puncturable PRF, this operation runs in time $s(n) \cdot \mathrm{poly}(\lambda, \log n)$.
- In the query phase, the client must search for a value $j \in [m]$ such that $i - \delta_j \in \mathsf{Eval}(\mathsf{sk})$. By the efficiency property of the puncturable pseudorandom set, computing the set $\mathsf{Eval}(\mathsf{sk})$ takes at most $s(n) \cdot \mathrm{poly}(\lambda, \log n)$. Using any data structure that supports fast lookups, the client can then find such a j, if one exists, in time $(s(n) + m(n)) \cdot \mathrm{poly}(\lambda, \log n)$.
- The client reconstruction procedure requires a simple addition and therefore satisfies the running time bound.

Having set $m(n) \coloneqq (n/s(n)) \log n$, the total running time of all three routines is $(s(n) + n/s(n)) \cdot \mathrm{poly}(\lambda, \log n)$.

The client's storage between the offline and online phases includes the key $\mathsf{sk} \in \mathcal{K}$ and the vectors $\delta \in [n]^m$ and $y \in \{0,1\}^m$. Their total length, for $m \coloneqq (n/s(n)) \log n$, is $\kappa + O((n/s(n)) \log^2 n)$.

Running λ instances of the scheme in parallel increases the storage and running time by a factor of λ. □

Claim (Security). *Assuming that the underlying puncturable pseudorandom set is secure, the offline/online PIR scheme is secure. More specifically, for every*

distinguishing adversary $\mathcal{A}$ attacking scheme Π_Ψ, there exists an adversary $\mathcal{B}$ attacking the underlying puncturable pseudorandom set Ψ such that for every $\lambda, n \in \mathbb{N}$, it holds that

$$\mathsf{PIRadv}[\mathcal{A}, \Pi](\lambda, n) \leq \mathrm{poly}(\lambda, n) \cdot \left(\mathsf{PSAdv}[\mathcal{B}, \Psi](\lambda, n) + 2^{-\lambda}\right) .$$

Proof. Recall that our scheme consists of λ instances of the fundamental scheme (for the purpose of amplifying its success probability), and when the client detects an error in all the instances, it retrieves the bit of interest using a non-private lookup.

If the puncturable pseudorandom set were perfectly secure, the failure probability of all λ instances would be at most $2^{-\lambda}$. By Proposition 34, each of the λ puncturable pseudorandom sets $\mathsf{Eval}(\mathsf{sk})$ (one at each instance) is computationally indistinguishable from a perfectly secure puncturable pseudorandom set. Therefore, by a standard hybrid argument, if the client fails with probability $\epsilon(\lambda, n)$ when using puncturable pseudorandom set Ψ, then a single instance of the client constitutes a distinguisher that has advantage at least $(\epsilon(\lambda, n) - 2^{-\lambda})/\lambda$ and, by the above efficiency analysis, runs in time $\widetilde{O}(n)$. Thus, by Proposition 34, there exists an efficient puncturable pseudorandom set adversary $\mathcal{B}_1$ such that

$$\epsilon(\lambda, n) \leq \lambda n s \cdot \mathsf{PSAdv}[\mathcal{B}_1, \Psi](\lambda, n) + 2^{-\lambda} . \tag{16}$$

Let $\mathcal{A}$ be an adversary attacking PIR scheme Π_Ψ. By a standard hybrid argument, there exists an adversary $\mathcal{A}'$ attacking a single instance Π' of the scheme such that

$$\mathsf{PIRadv}[\mathcal{A}, \Pi](\lambda, n) \leq \lambda \cdot \mathsf{PIRadv}[\mathcal{A}', \Pi'](\lambda, n) + \epsilon(\lambda, n) . \tag{17}$$

Let D_i be the distribution of the client's online query when reading index $i \in [n]$ *in a single instance of our scheme*. From the specification of the scheme in Construction 16, D_i is sampled as follows:

$\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)$
sample $\delta_1, \ldots, \delta_m \xleftarrow{\mathrm{R}} [n]$
sample a bit $b \xleftarrow{\mathrm{R}} \mathsf{Bernoulli}(\frac{s-1}{n})$
find a $j \in [m]$ such that $i - \delta_j \in \mathsf{Eval}(\mathsf{sk})$
if such a $j \in [m]$ exists:
  $\mathsf{sk}_q \leftarrow \mathsf{Shift}(\mathsf{sk}, \delta_j)$
otherwise:
  $i' \xleftarrow{\mathrm{R}} \mathsf{Eval}(\mathsf{sk})$
  $\mathsf{sk}_q \leftarrow \mathsf{Shift}(\mathsf{sk}, i - i')$
if $b = 0$: $i_{\mathsf{punc}} \leftarrow i$
else: $i_{\mathsf{punc}} \xleftarrow{\mathrm{R}} \mathsf{Eval}(\mathsf{sk}_q) \setminus \{i\}$
output $q \leftarrow \mathsf{Punc}(\mathsf{sk}_q, i_{\mathsf{punc}})$

We can rewrite this distribution as:

$\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)$
sample $\delta_1, \ldots, \delta_m \xleftarrow{\text{R}} [n]$
sample a bit $b \xleftarrow{\text{R}} \mathsf{Bernoulli}(\frac{s-1}{n})$
$i' \xleftarrow{\text{R}} \mathsf{Eval}(\mathsf{sk}) \cap \{i - \delta_1, \ldots, i - \delta_m\}$
if no such i' exists:
  $i' \xleftarrow{\text{R}} \mathsf{Eval}(\mathsf{sk})$
$\mathsf{sk}_q \leftarrow \mathsf{Shift}(\mathsf{sk}, i - i')$
if $b = 0$: $i_{\mathsf{punc}} \leftarrow i$
else: $i_{\mathsf{punc}} \xleftarrow{\text{R}} \mathsf{Eval}(\mathsf{sk}_q) \setminus \{i\}$
output $q \leftarrow \mathsf{Punc}(\mathsf{sk}_q, i_{\mathsf{punc}})$

Note that since the shifts $\delta_1, \ldots, \delta_m$ are a random, an element chosen from $\mathsf{Eval}(\mathsf{sk}) \cap \{i - \delta_1, \ldots, i - \delta_m\}$, whenever that intersection is not empty, is a uniformly random element in $\mathsf{Eval}(\mathsf{sk})$, and thus the distribution is identical to:

$\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)$
$i' \xleftarrow{\text{R}} \mathsf{Eval}(\mathsf{sk})$
$\mathsf{sk}_q \leftarrow \mathsf{Shift}(\mathsf{sk}, i - i')$
sample a bit $b \xleftarrow{\text{R}} \mathsf{Bernoulli}((s-1)/n)$
if $b = 0$: $i_{\mathsf{punc}} \leftarrow i$
else: $i_{\mathsf{punc}} \xleftarrow{\text{R}} \mathsf{Eval}(\mathsf{sk}_q) \setminus \{i\}$
output $q \leftarrow \mathsf{Punc}(\mathsf{sk}_q, i_{\mathsf{punc}})$

By definition of $\mathsf{GenWith}$, the last distribution is identical to the distribution $\mathcal{D}_{\lambda,n,i}$ in the statement of Lemma 36. Therefore, there exists an efficient adversary $\mathcal{B}_2$ such that

$$\mathsf{PIRadv}[\mathcal{A}', \Pi'](\lambda, n) \leq n^2 \cdot \mathsf{PSAdv}[\mathcal{B}_2, \Psi](\lambda, n)\,. \tag{18}$$

By Eqs. (16) to (18),

$$\begin{aligned}\mathsf{PIRadv}[\mathcal{A}, \Pi'] &\leq \lambda \cdot n^2 \cdot \mathsf{PSAdv}[\mathcal{B}_2, \Psi] + \epsilon \\ &\leq \lambda \cdot n^2 \cdot \mathsf{PSAdv}[\mathcal{B}_1, \Psi] + \lambda n s \cdot \mathsf{PSAdv}[\mathcal{B}_1, \Psi] + 2^{-\lambda}\,.\end{aligned}$$

Overall, if we construct an adversary $\mathcal{B}$, that runs each of $\mathcal{B}_1$ and $\mathcal{B}_2$ with probability $1/2$, we get that

$$\mathsf{PIRadv}[\mathcal{A}, \Pi](\lambda, n) \leq 4\lambda n^2 \cdot \mathsf{PSAdv}[\mathcal{B}, \Psi](\lambda, n) + 2^{-\lambda}\,. \qquad \square$$

D Additional material on the multi-query case (Section 4)

D.1 Definitions

Definition 40. A *multi-query* offline/online PIR scheme is a tuple of four polynomial-time algorithms:

- $\mathsf{Setup}(1^\lambda, n) \to (\mathsf{ck}, q_h)$, a randomized algorithm that takes in security parameter λ and database length n and outputs client key ck and a hint request q_h,

- $\mathsf{Hint}(x, q_h) \to h$, a deterministic algorithm that takes in a database $x \in \{0,1\}^n$ and a hint request q_h and outputs a hint h,
- $\mathsf{Query}(\mathsf{ck}, i) \to (\mathsf{ck}', q_{\mathsf{left}}, q_{\mathsf{right}})$, a randomized algorithm that takes as input the client's key ck and an index $i \in [n]$, and outputs an updated client key ck' and two queries,
- $\mathsf{Answer}^x(q) \to a$, a deterministic algorithm that takes as input a query q, and gets access to an oracle that:
  - takes as input an index $j \in [n]$, and
  - returns the j-th bit of the database $x_j \in \{0,1\}$,

  outputs an answer string a, and
- $\mathsf{Reconstruct}(h, a_{\mathsf{left}}, a_{\mathsf{right}}) \to (h', x_i)$, a deterministic algorithm that takes as input the hint and the answers from the two servers and outputs an updated hint h' and a database bit x_i.

A multi-query offline/online PIR scheme must satisfy the following notions of *correctness* and *security* that generalize those given in Definition 8 for the single-query case.

Correctness. We require that if the scheme is executed as prescribed, the client recovers the correct values of the bits at the retrieved indices.

More formally, for a multi-query offline/online PIR scheme $\Pi = (\mathsf{Setup}, \mathsf{Hint}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$, we require that for every $\lambda, n, T \in \mathbb{N}$ and every $i_1, \ldots, i_T \in [n]^T$, the following game always outputs "1":

- Compute $(\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n)$.
- Compute $h \leftarrow \mathsf{Hint}(x, q_h)$.
- For $t = 1, \ldots, T$, compute:

$$\begin{aligned}(\mathsf{ck}, q_{\mathsf{left}}, q_{\mathsf{right}}) &\leftarrow \mathsf{Query}(\mathsf{ck}, i_t)\\ a_{\mathsf{left}} &\leftarrow \mathsf{Answer}^x(q_{\mathsf{left}})\\ a_{\mathsf{right}} &\leftarrow \mathsf{Answer}^x(q_{\mathsf{right}})\\ (h, x'_{i_t}) &\leftarrow \mathsf{Reconstruct}(h, a_{\mathsf{left}}, a_{\mathsf{right}})\end{aligned}$$

- Output "1" if $x'_{i_t} = x_{i_t}$ for every $t \in [T]$ and "0" otherwise.

Security. Informally, we require that each server on its own "learns nothing" about the indices read by the client, even if the server can adaptively choose those indices.

Formally, for a multi-query offline/online PIR scheme $\Pi = (\mathsf{Setup}, \mathsf{Hint}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ and a *stateful* algorithm $\mathcal{A}$, we define two games (Games 41 and 42), parametrized by security parameter λ, database size $n \in \mathbb{N}$, number of queries $T \in \mathbb{N}$, and a bit $b \in \{0,1\}$.

Let $W^{\mathsf{left}}_{\lambda,n,T,b}$ be the event that the adversary outputs $b' = 1$ in the left-server game (Game 41), instantiated with parameters $\lambda, n, T \in \mathbb{N}$ and $b \in \{0,1\}$. Let $W^{\mathsf{right}}_{\lambda,n,T,b}$ be the same event in the right-server game (Game 42). We define the advantage of adversary $\mathcal{A}$:

$$\mathsf{PIRadv}[\mathcal{A}, \Pi](\lambda, n, T) \coloneqq \max_{\sigma \in \{\mathsf{right},\mathsf{right}\}} \left|\Pr[W^\sigma_{\lambda,n,T,0}] - \Pr[W^\sigma_{\lambda,n,T,1}]\right| .$$

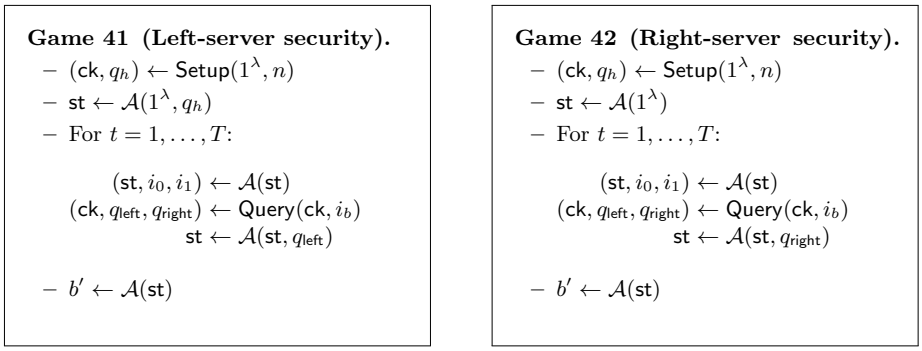

Fig. 3: Security games for multi-query private information retrieval.

We say that a multi-query offline/online PIR scheme Π is *secure* if for all efficient stateful algorithms $\mathcal{A}$, all polynomially bounded functions $n(\lambda)$ and $T(\lambda)$, and all values of $\lambda \in \mathbb{N}$, $\mathsf{PIRadv}[\mathcal{A}, \Pi](\lambda, n(\lambda), T(\lambda))$ is a negligible function of λ.

Remark 43 (Malicious security). The client's queries depend neither on the hint nor on any of the servers' past answers. For this reason, the hint and the servers' answers do not play a role in the security games we define. Thus, actively malicious behavior by the servers, such as responding incorrectly to a client query, cannot help them break the PIR security property.

Efficiency. We consider the following measures of efficiency. All of the following are measured as a function of n and λ, and we use their worst case values over the choice of the database $x \in [n]$, read indices, and all randomness used by the servers.

- The offline communication is $|q_h| + |h|$.
- The online communication is $|q_{\mathsf{right}}| + |q_{\mathsf{left}}| + |a_{\mathsf{left}}| + |a_{\mathsf{right}}|$.
- The offline running time is the running time of the routine Hint.
- The online server running time is sum of the running times of Answer over both $q_{\mathsf{left}}, q_{\mathsf{right}}$.
- The client's running time is the sum of the running times of Query and $\mathsf{Reconstruct}$.

All of the above should be polynomial in λ, and ideally all, except the offline running time, should be sublinear in the size of the database.

D.2 Proof of Theorem 17

Let $\Pi = (\mathsf{Setup}, \mathsf{Hint}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ be the multi-query offline/online PIR scheme of Construction 44. The scheme is parametrized by a positive integer $s \leq n/2$, which we later set to $s := \sqrt{n}$. The scheme uses a puncturable pseudorandom set with a universe of size n, set size s, key space $\mathcal{K}$, and punctured-key space $\mathcal{K}_\mathsf{p}$. We also let $m := (2n/s) \log n$.

As presented, the scheme fails with non-negligible probability (on failure, the query algorithm outputs $\perp$). We call this a single instance of our scheme. The final scheme consists of running λ instances in parallel using independent randomness. For each query, the client queries all instances in parallel, and obtain the correct answer as long as at least one of the instances succeeds.

The following lemma will be useful in proving both correctness and security. Informally, we show that in the multi-query PIR scheme, each query to the right server completely hides from the right server both the retrieved index and the client's updated key. Moreover, the client's updated key is distributed identically to the initial client key. This guarantees that, as far as the right server is concerned, the client goes back to a "clean state" after each query. This will later be instrumental in proving security of the multi-query scheme.

Lemma 45. *Let $\Pi = (\mathsf{Setup}, \mathsf{Hint}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ be the multi-query offline/online PIR scheme of Construction 44 and suppose that the underlying puncturable pseudorandom set Ψ is secure. Then, for every polynomially bounded $n = n(\lambda)$, every $x \in \{0,1\}^n$, and every $i_0, i_1 \in [n]$,*

$$\left\{\begin{array}{c}(\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n)\\ (\mathsf{ck}, q_{\mathsf{left}}, q_{\mathsf{right}}) \leftarrow \mathsf{Query}(\mathsf{ck}, i_0)\\ \text{output } (q_{\mathsf{right}}, \mathsf{ck})\end{array}\right\} \approx_c \left\{\begin{array}{c}\mathsf{ck}, q_h \leftarrow \mathsf{Setup}(1^\lambda, n)\\ (\mathsf{ck}, q_{\mathsf{left}}, q_{\mathsf{right}}) \leftarrow \mathsf{Query}(\mathsf{ck}, i_1)\\ (\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n)\\ \text{output } (q_{\mathsf{right}}, \mathsf{ck})\end{array}\right\}.$$

Furthermore, if the puncturable pseudorandom set Ψ is ϵ_Ψ-secure, then the maximal advantage of any efficient adversary at distinguishing these distributions is $\epsilon(\lambda, n) \leq \mathrm{poly}(n) \cdot \epsilon_\Psi(\lambda, n)$.

Proof. Let $\Psi = (\mathsf{Gen}, \mathsf{Punc}, \mathsf{Eval})$ be the underlying puncturable pseudorandom set with set size s, key space $\mathcal{K}$ and punctured-key space $\mathcal{K}_\mathsf{p}$.

Let $q_{\mathsf{right}}(i)$ be the distribution of the right-server query after reading a single index i using the scheme Π. Furthermore, let $\mathsf{ck}(i)$ be the distribution of the client key after reading index i. We thus need to show the following:

- the updated key $\mathsf{ck}(i_0)$ is indistinguishable from m set keys sampled using $\mathsf{Gen}(1^\lambda, n)$ independently of each other and independently of q_{right}, and
- the query $q_{\mathsf{right}}(i_0)$ is indistinguishable from $q_{\mathsf{right}}(i_1)$.

We start by writing the first distribution above explicitly.

Distribution 0. Generate $q_{\mathsf{right}} \in \mathcal{K}_\mathsf{p}$ and $\mathsf{ck} \in \mathcal{K}^m$ as follows:

$(\mathsf{sk}_1, \ldots, \mathsf{sk}_m) \leftarrow (\mathsf{Gen}(1^\lambda, n), \ldots, \mathsf{Gen}(1^\lambda, n))$
$\mathsf{sk}_{\mathsf{new}} \leftarrow \mathsf{GenWith}(1^\lambda, n, i_0)$
sample a bit $b \xleftarrow{\mathrm{R}} \mathsf{Bernoulli}((s-1)/n)$
if $i_0 \in \cup_{j=1}^m \mathsf{Eval}(\mathsf{sk}_j)$ and $b = 0$ then:
  let $j \in [m]$ be such that $i_{\mathsf{pir}} \in \mathsf{Eval}(\mathsf{sk}_j)$
  swap sk_j and $\mathsf{sk}_{\mathsf{new}}$
$S \leftarrow \mathsf{Eval}(\mathsf{sk}_{\mathsf{new}})$
if $b = 0$ then $i_{\mathsf{punc}} \leftarrow i_0$ else $i_{\mathsf{punc}} \xleftarrow{\mathrm{R}} S \setminus \{i_0\}$
output $q_{\mathsf{right}} \leftarrow \mathsf{Punc}(\mathsf{sk}_{\mathsf{new}}, i_{\mathsf{punc}})$ and $\mathsf{ck} \leftarrow (\mathsf{sk}_1, \ldots, \mathsf{sk}_m)$

Construction 44 (Multi-query offline/online PIR). The construction is parametrized by set size $s\colon \mathbb{N} \to \mathbb{N}$ and uses a puncturable pseudorandom set $\Psi = (\mathsf{Gen}, \mathsf{Punc}, \mathsf{Eval})$ with set size s, extended with routine $\mathsf{GenWith}$. The following is a single instance of the scheme. The final scheme is obtained by running λ instances in parallel.
Throughout, we let $m := (2n/s)\log n$.

Offline phase

$\underline{\mathsf{Setup}(1^\lambda, n) \to (\mathsf{ck}, q_h)}$

for $j = 1, \ldots, m$ do:
  $\mathsf{sk}_j \leftarrow \mathsf{Gen}(1^\lambda, n)$
$\mathsf{ck} \leftarrow (\mathsf{sk}_1, \ldots, \mathsf{sk}_m)$
output ck and $q_h \leftarrow \mathsf{ck}$

$\underline{\mathsf{Hint}(q_h,\ x \in \{0,1\}^n) \to h \in \{0,1\}^m}$

parse q_h as $\mathsf{sk}_1 \ldots, \mathsf{sk}_m \in \mathcal{K}$
for $j = 1, \ldots, m$ do:
  $S_j \leftarrow \mathsf{Eval}(\mathsf{sk}_j)$
  $h_j \leftarrow \sum_{i \in S_j} x_i \bmod 2$
output $h \leftarrow (h_1, \ldots, h_m)$

Online phase

$\underline{\mathsf{Query}(\mathsf{ck}, i) \to (\mathsf{ck}, q_{\mathsf{left}}, q_{\mathsf{right}})}$

$\mathsf{sk}_{\mathsf{new}} \leftarrow \mathsf{GenWith}(1^\lambda, n, i)$
sample a bit $b \xleftarrow{\mathrm{R}} \mathsf{Bernoulli}((s-1)/n)$
if $i \in \cup_{j=1}^m \mathsf{Eval}(\mathsf{sk}_j)$ and $b = 0$ then:
  let $j \in [m]$ be such that $i \in \mathsf{Eval}(\mathsf{sk}_j)$
  $\mathsf{sk}_{\mathsf{right}} \leftarrow \mathsf{sk}_j$
  update ck with $\mathsf{sk}_j \leftarrow \mathsf{sk}_{\mathsf{new}}$
else:
  $j \leftarrow \perp$
  $\mathsf{sk}_{\mathsf{right}} \leftarrow \mathsf{sk}_{\mathsf{new}}$
if $b = 0$ then $i_{\mathsf{punc}} \leftarrow i$
else $i_{\mathsf{punc}} \xleftarrow{\mathrm{R}} \mathsf{Eval}(\mathsf{sk}_{\mathsf{new}}) \setminus \{i\}$
$q_{\mathsf{left}} \leftarrow \mathsf{Punc}(\mathsf{sk}_{\mathsf{new}}, i_{\mathsf{punc}})$
$q_{\mathsf{right}} \leftarrow \mathsf{Punc}(\mathsf{sk}_{\mathsf{right}}, i_{\mathsf{punc}})$
output ck, q_{left}, and q_{right}

$\underline{\mathsf{Reconstruct}(h, a_{\mathsf{left}}, a_{\mathsf{right}}) \to (h', x_i)}$

let $j \in [m]$ be as in $\mathsf{Query}^\dagger$
if $j = \perp$ output $\perp$
$x_i \leftarrow h_j - a_{\mathsf{right}} \bmod 2$
update $h_j \leftarrow a_{\mathsf{left}} + x_i \bmod 2$
output the updated h and x_i

$\underline{\mathsf{Answer}^x(q) \to a}$

$S \leftarrow \mathsf{Eval}(q)$
return $a \leftarrow \sum_{i \in S} x_i \bmod 2$

Client's storage
key $\mathsf{ck} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_m) \in \mathcal{K}^m$
hint $h \in \{0,1\}^m$

†For simplicity, we avoid passing j explicitly from Query to $\mathsf{Reconstruct}$.

By Construction 44, Distribution 0 is identical to the first distribution in the statement of the lemma.

Distribution 1. We observe that if $i_0 \in \mathsf{sk}_j$ in Distribution 0, then $\mathsf{sk}_{\mathsf{new}}$ and sk_j are identically distributed. Both are also independent of $\mathsf{sk}_{j'}$ for every other $j' \neq j$, and therefore the joint distributions of $\mathsf{sk}_1, \ldots, \mathsf{sk}_m$ and $\mathsf{sk}_1, \ldots, \mathsf{sk}_{j-1}, \mathsf{sk}_{\mathsf{new}}, \mathsf{sk}_{j+1}, \ldots, \mathsf{sk}_m$ are identical. We can therefore remove the swap between $\mathsf{sk}_{\mathsf{new}}$ and sk_j in the marked line above, without changing the output distribution of the challenger.

$(\mathsf{sk}_1, \ldots, \mathsf{sk}_m) \leftarrow (\mathsf{Gen}(1^\lambda, n), \ldots, \mathsf{Gen}(1^\lambda, n))$
$\mathsf{sk}_{\mathsf{new}} \leftarrow \mathsf{GenWith}(1^\lambda, n, i_0)$
sample a bit $b \xleftarrow{\mathrm{R}} \mathsf{Bernoulli}((s-1)/n)$
$S \leftarrow \mathsf{Eval}(\mathsf{sk}_{\mathsf{new}})$
if $b = 0$ then $i_{\mathsf{punc}} \leftarrow i_0$ else $i_{\mathsf{punc}} \xleftarrow{\mathrm{R}} S \setminus \{i_0\}$
output $q_{\mathsf{right}} \leftarrow \mathsf{Punc}(\mathsf{sk}_{\mathsf{new}}, i_{\mathsf{punc}})$ and $\mathsf{ck} \leftarrow (\mathsf{sk}_1, \ldots, \mathsf{sk}_m)$

The resulting distribution is therefore (statistically) identical to Distribution 0. From here, we obtain the first part of the lemma, namely that the updated set key ck consists of m random set keys, which are independent of each other and of q_{right}.

Distribution 2. Generate $q_{\mathsf{right}} \in \mathcal{K}_{\mathsf{p}}$ as follows:

$\mathsf{sk}_{\mathsf{new}} \leftarrow \mathsf{GenWith}(1^\lambda, n, i_0)$
sample a bit $b \xleftarrow{\mathrm{R}} \mathsf{Bernoulli}((s-1)/n)$
$S \leftarrow \mathsf{Eval}(\mathsf{sk}_{\mathsf{new}})$
if $b = 0$ then $i_{\mathsf{punc}} \leftarrow i_0$ else $i_{\mathsf{punc}} \xleftarrow{\mathrm{R}} S \setminus \{i_0\}$
output $q_{\mathsf{right}} \leftarrow \mathsf{Punc}(\mathsf{sk}_{\mathsf{new}}, i_{\mathsf{punc}})$

Since in Distribution 1, q_{right} is independent from ck, the marginal distribution of q_{right} in Distribution 1 is identical to Distribution 2. By Lemma 36, in Distribution 2, the query $q_{\mathsf{right}}(i_0)$ is indistinguishable from $q_{\mathsf{right}}(i_1)$ for every $i_0, i_1 \in [n]$. Furthermore, Lemma 36 bounds the distinguishing advantage as well, and the lemma follows. □

Correctness. We first analyze the failure probability of a single instance of our scheme. To this end, observe that reading a bit $i \in [n]$ fails when $i \notin \cup_{j=1}^m \mathsf{Eval}(\mathsf{sk}_j)$ or $i_{\mathsf{punc}} \neq i$. Initially, the offline phase sets $\mathsf{ck} \leftarrow (\mathsf{Gen}(1^\lambda, n), \ldots, \mathsf{Gen}(1^\lambda, n)) \in \mathcal{K}^m$. By Lemma 45, if, *before a query*, the client's key contains m independent random set keys generated by Gen, then so does it *after the next query*. Therefore, by induction, the client's key consists of m independent random set keys throughout. If the puncturable pseudorandom set were perfectly secure, we would have had:

$$\Pr\left[\exists\, i \notin \cup_{j=1}^m \mathsf{Eval}(\mathsf{sk}_j)\right] \leq \sum_{i=1}^{n} \Pr\left[i \notin \mathsf{Eval}(\mathsf{sk}) : \mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda, n)\right]^m \leq n(1-\tfrac{s}{n})^m$$

For $m = (2n/s)\log n$, we have seen in Eq. (15) that this is at most $1/n$.

The second failure event, $i_{\mathsf{punc}} \neq i$, occurs when $c = 1$, the probability of which is $(s-1)/n$.

Therefore, when using *perfectly secure puncturable pseudorandom sets*, each read fails with probability at most $s/n \leq 1/2$ for $s \leq n/2$. When running λ instances of our scheme in parallel, the probability that each single read fails across all λ schemes is then $2^{-\lambda}$. Taking the union bound over all T reads results in a failure probability of at most $T/2^{\lambda}$.

Now let $\epsilon(\lambda, n, T)$ be the probability that the scheme, instantiated using a puncturable pseudorandom set Ψ, fails to read, from a database of size n, any bit in an adaptively-chosen sequence of T indices. Proposition 34 shows that each of the $m\lambda T$ puncturable pseudorandom sets (m per instance per operation) is computationally indistinguishable from a perfectly secure puncturable pseudorandom set. Therefore, by a standard hybrid argument, there exists an adversary $\mathcal{A}$ that attacks the puncturable pseudorandom set Ψ such that

$$\epsilon(\lambda, n, T) \leq \lambda m T \cdot \mathsf{PSAdv}[\mathcal{A}, \Psi](\lambda, n) + 2^{-\lambda}\,. \tag{19}$$

As in the proof of Lemma 15, it is now possible to trade the failure probability for a negligible loss in security and get a scheme with perfect correctness.

Efficiency. The offline communication for a single instance consists of the server sending r and h to the client. Their combined length is $m \log |\mathcal{K}| + m$ bits. When running λ instances in parallel, the cost increases λ-fold. However, as an optimization, the left server can generate all λm set keys using a pseudorandom generator, and send one λ-bit seed instead of the $\lambda \cdot m$ secret keys, which brings the communication in the offline phase down to $O(m\lambda)$ instead of $O(m\lambda^2)$. Setting $m = (2n/s)\log n = 2\sqrt{n}\log n$, we get that the total communication in the offline phase of our scheme is $O(\lambda\sqrt{n}\log n)$.

Using the puncturable pseudorandom set construction of Theorem 7 gives online communication $O(\lambda\sqrt{n}\log n)$ bits and a running time at the server that matches that of the single-query scheme of Theorem 14.

The client stores $m = \widetilde{O}_\lambda(n^{1/2})$ puncturable pseudorandom set keys, each of length λ bits, for total storage $\widetilde{O}_\lambda(n^{1/2})$.

The bulk of the client's online work consists of finding a set key sk_j in the list $(\mathsf{sk}_1, \ldots, \mathsf{sk}_m)$ such that its desired index $i \in \mathsf{Eval}(\mathsf{sk}_j)$. Since the puncturable pseudorandom set of Theorem 7 supports fast membership lookups, for every $j \in [m]$, the client can test whether $i \in \mathsf{Eval}(\mathsf{sk}_j)$ in time $\mathrm{poly}(\lambda, \log n)$. Since $m = \widetilde{O}(n^{1/2})$, this gives a total client online time of $\widetilde{O}_\lambda(n^{1/2})$.

Left-server security. We show that for every polynomially bounded database length $n(\lambda)$ and every (adaptively chosen) polynomially long sequence of read indices, the queries to the left server are indistinguishable from random punctured set keys.

From the specification of the query algorithm in Construction 44, the left-server query is generated as follows:

$\mathsf{sk}_{\mathsf{new}} \leftarrow \mathsf{GenWith}(1^\lambda, n, i)$
sample a bit $b \xleftarrow{\text{R}} \mathsf{Bernoulli}((s-1)/n)$
if $b = 0$ then $i_{\mathsf{punc}} \leftarrow i$
else $i_{\mathsf{punc}} \xleftarrow{\text{R}} \mathsf{Eval}(\mathsf{sk}_{\mathsf{new}}) \setminus \{i\}$
$q_{\mathsf{left}} \leftarrow \mathsf{Punc}(\mathsf{sk}_{\mathsf{new}}, i_{\mathsf{punc}})$

Therefore, the queries to the left server are independent of the client's key and of any past queries. It is therefore sufficient to consider a single query in isolation. By Lemma 36, for every $i, j \in [n]$, the left query $q_{\mathsf{left}}(i)$ when reading index i is indistinguishable from $q_{\mathsf{left}}(j)$ when reading index j, and the left-server security follows.

Right-server security. We prove security by using Lemma 45 together with a standard hybrid argument.

As in the single-query case, it is sufficient to show security for *a single instance* of our scheme, though now we need to show security for a sequence of adaptive queries. Let $\mathcal{A}$ be a stateful adversary that attacks, as in Game 42, a single instance Π' of the PIR scheme. For $\lambda, n, T \in \mathbb{N}$, consider then the following sequence of $T+1$ hybrid experiments:

Experiment $t = 0, 1, \ldots, T$

$(\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n)$
$\mathsf{st} \leftarrow \mathcal{A}(1^\lambda, q_h)$
for $\tau = 1$ to t:
  $(\mathsf{st}, i_0, i_1) \leftarrow \mathcal{A}(\mathsf{st})$
  $(\mathsf{ck}, q_{\mathsf{left}}, q_{\mathsf{right}}) \leftarrow \mathsf{Query}(\mathsf{ck}, i_0)$
  $\mathsf{st} \leftarrow \mathcal{A}(\mathsf{st}, q_{\mathsf{right}})$
// Wipe client key
$(\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n)$
for $\tau = t+1$ to T:
  $(\mathsf{st}, i_0, i_1) \leftarrow \mathcal{A}(\mathsf{st})$
  $(\mathsf{ck}, q_{\mathsf{left}}, q_{\mathsf{right}}) \leftarrow \mathsf{Query}(\mathsf{ck}, i_1)$
  $\mathsf{st} \leftarrow \mathcal{A}(\mathsf{st}, q_{\mathsf{right}})$
$b' \leftarrow \mathcal{A}(\mathsf{st})$

Experiment 0 corresponds to the case of $b = 0$ in Game 42. Furthermore, Experiment $T+1$ corresponds to the case $b = 1$ in Game 42. Consider now two adjacent games in the sequence:

- In Experiment t, right query number t is obtained by running the PIR client on index i_0, and the client's updated key carries over to the next query.
- In Experiment $t+1$, right query number t is obtained by running the PIR client on index i_1, and the client's updated key for the next query is generated from scratch.
- All other queries are identical between two neighboring experiments.

By Lemma 45, when n is a polynomially bounded function of λ, these neighboring games are computationally indistinguishable. Furthermore, when T is

polynomially bounded, the first and the last hybrid are then indistinguishable as well, and if the underlying puncturable pseudorandom set it ϵ_Ψ-secure, we have that

$$\mathsf{PIRadv}[\mathcal{A}, \Pi'](\lambda, n) \leq \mathrm{poly}(\lambda, n) \cdot \epsilon_\Psi(\lambda, n)\,.$$

Having established correctness, efficiency, and security, Theorem 17 follows.

D.3 Proof of Corollary 18

The corollary follows by:

1. instantiating Construction 44 with the puncturable pseudorandom set construction based on puncturable PRFs (Theorem 3) to reduce the online communication, and
2. using a preprocessed data structure to minimize the client's storage and online time.

Instantiating Construction 44 with the puncturable set construction of Theorem 3, based on puncturable PRFs, immediately reduces the online communication cost to $O(\lambda^2 \log n)$, following a similar analysis to the one used in the single-query case (Theorem 14).

The challenge is that the puncturable pseudorandom set construction based on puncturable PRFs does *not* support a fast membership test. That is, given a set key sk, the fastest way we know to test whether $i \in \mathsf{Eval}(\mathsf{sk})$ is to compute $S \leftarrow \mathsf{Eval}(\mathsf{sk})$ explicitly, in time $s(n) \cdot \mathrm{poly}(\lambda, \log n)$, and check whether $i \in S$.

In the online phase, the client has an index $i \in [n]$ and a list of set keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_m)$ in its storage. The client must find a set key sk_j in this list such that $i \in \mathsf{Eval}(\mathsf{sk}_j)$.

When using a puncturable pseudorandom set *without* a fast membership test, the client can achieve storage and online time $\widetilde{O}_\lambda(n^{5/6})$ by using use a classic data structure for time-space trade-offs [Hel80, FN00].

In particular, for each set key $\mathsf{sk}_j \in \{\mathsf{sk}_1, \ldots, \mathsf{sk}_m\}$, the client defines the function $f_j : [s(n)] \to [n]$, where

$$f_j(\ell) = \text{“the } \ell\text{th element of the set } \mathsf{Eval}(\mathsf{sk}_\ell)\text{”} \subseteq [n].$$

When using the PRF-based puncturable pseudorandom set construction, computing the value of f_j on an individual point does not require evaluating the entire size-s set, and in particular we can compute individual values of f_j in time $\mathrm{poly}(\lambda, \log n)$.

The client then builds a Hellman-table data structure for inverting each of the functions $f_1, \ldots, f_m$. (The inversion algorithms of Hellman and of Fiat and Naor typically apply to length-preserving functions, but a straightforward application of hashing gives the same complexity for random injective length-increasing functions, as we have here.)

Each of these m tables takes time $\widetilde{O}_\lambda(s(n)) = \widetilde{O}_\lambda(\sqrt{n})$ to construct, with $s(n) = O(\sqrt{n})$, for a total offline time of $\widetilde{O}_\lambda(m\sqrt{n}) = \widetilde{O}_\lambda(n)$. The space required for each table is $s(n)^{2/3}$ bits, for a total space usage of $m \cdot \widetilde{O}_\lambda(s(n)^{2/3}) = \widetilde{O}_\lambda(n^{5/6})$ bits.

In the online phase, given a point $i \in [n]$, if the client can find an $j \in [m]$ and $\ell \in [s(n)]$ such that $f_j(\ell) = i$, then the client knows that $i \in \mathsf{Eval}(\mathsf{sk}_j)$. Thus, in the online phase, the client uses its precomputed tables to try to find the inverse of i under each of the functions $f_1, f_2, \ldots, f_m$. Each of these m searches takes time $\widetilde{O}_\lambda(s(n)^{2/3})$, for a total time of $\widetilde{O}_\lambda(n^{5/6})$.

The client can update this data structure (i.e., replace one key sk_j with a fresh one $\hat{\mathsf{sk}}_j$) in the time it takes to create a single Hellman table, which is only $\widetilde{O}_\lambda(n^{1/2})$.

E Additional material on the single-server case (Section 5)

E.1 Definitions

Definition 46 (Single-server offline/online PIR). A single-server offline/online PIR scheme is an offline/online PIR scheme, as in Definition 8 that satisfies the following additional security property:

Single-server security. For a security parameter $\lambda \in \mathbb{N}$, database length $n \in \mathbb{N}$, and index $i \in [n]$, define the distribution

$$D_{i,\lambda,n} := \left\{ (q_h, q) : \begin{array}{c} (\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n) \\ q \leftarrow \mathsf{Query}(\mathsf{ck}, i) \end{array} \right\} .$$

We say that an offline/online PIR scheme is *single-server secure* if, for every polynomially bounded $n = n(\lambda)$, and $i, j \in [n]$, the ensembles $\{D_{\lambda,n,i}\}_{\lambda\in\mathbb{N}}$ and $\{D_{\lambda,n,j}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable.

Security analysis of the scheme in Theorem 20. The security follows from the security of the underlying primitives (linearly homomorphic encryption, single-server PIR, and two-server offline/online PIR). More formally, when a client queries index $i \in [n]$, the view $\mathsf{View}(i)$ of the server consists of:

- an encryption of the indicator vector of the set S,
- the query string for the batch PIR algorithm, retrieving indices $(\delta_1, \ldots, \delta_m)$, and
- the client's online query for index $i \in [n]$ using the two-server PIR scheme of Theorem 14.

A hybrid argument shows that for any pair of indices $i, j \in [n]$, views $\mathsf{View}(i) \approx_c \mathsf{View}(j)$. The steps of the hybrid are:

1. swap the set S for an empty set, using the semantic security of the encryption scheme,
2. swap the indices $(\delta_1, \ldots, \delta_m)$ for a fixed string, using the security of the single-server PIR scheme,
3. swap the online query for i for an online query for j, using the security of the two-server scheme, and
4. swap the δs and encryption back, again using the single-server PIR security and the security of the encryption scheme.

E.2 A simple single-server scheme

We sketch a simple variant of an offline/online single-server PIR scheme with linear online time. This simple scheme can be obtained from Theorem 20, but we construct it explicitly for ease of exposition.

We obtain this variant by transforming one of the two-server PIR schemes from the original PIR paper [CGKS98] into a single-server offline/online PIR, using linearly homomorphic encryption.

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a linearly homomorphic encryption scheme over message space $\mathbb{F}$ and ciphertext space $\mathbb{G}$. (The security does not depend on the field $\mathbb{F}$ being large—it could even be $\mathbb{F}_2$.)

The client and server represent the n-bit database as a bit-matrix X of dimensions $\sqrt{n} \times \sqrt{n}$.

– **Offline phase.**

- On security parameter $\lambda \in \mathbb{N}$, the client samples a random vector $v \xleftarrow{R} \mathbb{F}^{\sqrt{n}}$ and an encryption key $k \leftarrow \mathsf{Gen}(1^\lambda)$, computes the component-wise encryption $\mathsf{ct}_v \leftarrow \mathsf{Enc}(k, v) \in \mathbb{G}^{\sqrt{n}}$ of the vector v, and sends ct_v to the server.
- The server computes under encryption the vector-matrix product $h = v^T X \in \mathbb{F}^{\sqrt{n}}$, where X is the database. Here we use the linear homomorphism of the encryption scheme. That is, the server computes

$$\mathsf{ct}_h = \mathsf{Enc}(k, v^T) \cdot X = \mathsf{Enc}(k, v^T X) = \mathsf{Enc}(k, h) \quad \in \mathbb{F}^{\sqrt{n}}$$

  and returns ct_h to the client.
- The client computes $h \leftarrow \mathsf{Dec}(k, \mathsf{ct}_h) \in \mathbb{F}^{\sqrt{n}}$, and stores $(v, h) \in \mathbb{F}^{\sqrt{n}} \times \mathbb{F}^{\sqrt{n}}$ as its hint.

– **Online phase.** In the online phase, the client has an index $(i, j) \in [\sqrt{n}]^2$ and it wants to read the bit X_{ij} from the database. For any $i \in [\sqrt{n}]$, let $e_i \in \mathbb{F}^{\sqrt{n}}$ be the vector of all zeros except with a "1" at coordinate i.

- The client computes $v' \leftarrow e_i - v \in \mathbb{F}^{\sqrt{n}}$ and sends v' *unencrypted* to the server.
- The server computes the answer $a \leftarrow (v')^T X \in \mathbb{F}^{\sqrt{n}}$ and returns a to the client.
- The client computes the ith row of the database X as:

$$X_i = h + a = v^T X + (v')^T X = v^T X + (e_i - v)^T X = e_i X \quad \in \mathbb{F}^{\sqrt{n}}.$$

  The client then can output $X_i e_j = x_{ij} \in \{0, 1\} \subseteq \mathbb{F}$.

The total communication consists of $2\sqrt{n}$ ciphertexts in the offline phase and $2\sqrt{n}$ bits in the online phase. The offline computation requires n homomorphic operations at the server, and the online phase requires $O(n)$ bit operations at the server.

More generally, this approach allows obtaining a single-server offline/online PIR scheme from any information-theoretic two-server PIR schemes in which the answer of one of the two servers is a linear function of the client's query.

E.3 Discussion and extensions

Further reducing online communication. By combining the construction of Theorem 20 with a standard single-server PIR scheme [KO97, CMS99, KO00, GR05, OS07], we can reduce the online-phase communication at the cost of requiring public-key cryptographic operations in the online phase. Still, the number of public-key operations required in the online phase is *sublinear* in the database size, so the online phase will still be aysmptotically (and probably also concretely) faster than just using standard (online-only) single-server PIR.

To sketch the idea: In the online phase of the construction above, the server sends the client $n^{1/3}$ ciphertexts, but the clients only needs one of them. Instead, the client can use standard single-server PIR to privately retrieve the single ciphertext that it actually needs. (This is very much in the spirit of the original single-server PIR paper of Kushilevitz and Ostrovsky [KO97].)

A square-root scheme without FHE? It would be excellent to construct a single-server offline/online PIR scheme with the efficiency properties of this FHE-based scheme, but under weaker assumptions (e.g., linearly homomorphic encryption, single-server PIR, or assumptions on bilinear groups). The question is interesting even if we allow the server to run in super-linear time in the offline phase.

One possible direction would be to construct a *PIR-for-parities* scheme. In a PIR for parities scheme, the server holds a string $x \in \{0,1\}^n$, and the client holds a set $S \subseteq [n]$ of size s. The client wants to learn the value $\sum_{i \in S} x_i \bmod 2$ without leaking its set S to the server. Is it possible to construct a single-server PIR-for-parities scheme in which the client uploads $s \cdot \mathrm{poly}(\lambda, \log n)$ bits to the server and downloads $\mathrm{poly}(\lambda, \log n)$ bits from the server? Plugging such a scheme into the construction of Theorem 20 would give a single-server offline/online PIR scheme that achieves the optimal $\widetilde{O}_\lambda(n^{1/2})$ offline communication and online time.

Theorem 20 implicitly constructs such a PIR-for-parities scheme using linearly homomorphic encryption that achieves $\widetilde{\Omega}_\lambda(n)$ upload and $\widetilde{O}_\lambda(1)$ download. Using a result of Ishai and Paskin [IP07], we can get a scheme from simple assumptions [DGI+19] with $s \cdot \mathrm{poly}(\lambda, \log n)$ upload *and* download, but we know of no way to achieve order s upload and order 1 download from simple assumptions.

F Proof of Theorem 23 (the lower bound)

To prove Theorem 23, we show that any offline/online PIR scheme implies an algorithm for the following oracle problem, first studied by Yao [Yao90]. We first sketch the problem informally, and then give a formal definition.

Informally, the Box Problem is a two-player game with two phases, with each player playing in a different phase:

- **Preprocessing phase.** The first player is given access to an oracle $\mathcal{O} : [n] \to \{0,1\}$. She may make an unbounded number of queries to the oracle and then may write down a C-bit advice string $\mathsf{st}_{\mathcal{O}}$ about the oracle.

– **Online phase.** The second player is given the precomputed string $\mathsf{st}_{\mathcal{O}}$ and a point $i \in [n]$. She then may make at most T queries to the oracle, provided that she does *not* query the oracle at the point i. At the end, she must output a value $y' \in \{0,1\}$, such that $y' = \mathcal{O}(i)$.

The players may coordinate a common strategy ahead of time.

We can formalize the task as follows:

Definition 47 (Yao's Box Problem [Yao90]). We say that a pair of oracle algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ ϵ-solves Yao's Box Problem on parameter n using C bits of advice and T queries if the pair of algorithms satisfies the following properties:

– $\mathcal{A}_1$ outputs an advice string of at most C bits,
– $\mathcal{A}_2$ makes at most T oracle queries,
– for all $i \in [n]$,

$$\Pr_{\mathcal{A},\mathcal{O},i}\left[\mathcal{A}_2^{\mathcal{O}}(\mathcal{A}_1^{\mathcal{O}}(), i) = \mathcal{O}(i)\right] \geq 1/2 + \epsilon\,,$$

and
– $\mathcal{A}_2^{\mathcal{O}}(\cdot, i)$ never queries its oracle on i.

Yao [Yao90] proved the following theorem for $\epsilon = 1$, and the generalization to the sub-constant error regime follows by either a compression argument [GT00, DTT10, DGK17] or by presampling methods [Unr07, CDGS18].

Theorem 48 ([Yao90]). *Any algorithm that ϵ-solves Yao's Box Problem using C bits of advice and T queries satisfies $\epsilon = \widetilde{O}\left(\sqrt{C(T+1)/N}\right)$.*

Before continuing to the proof of Theorem 23, we prove the following lemma about secure PIR schemes, which we will need for the main proof.

Lemma 49. *Let $\Pi = (\mathsf{Setup}, \mathsf{Hint}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ be an offline/online PIR scheme. For security parameter $\lambda \in \mathbb{N}$, database size $n \in \mathbb{N}$, and query index $i \in [n]$, let the probability distribution D_i be as in Eq. (3):*

$$D_i := \left\{ q : \begin{array}{c} (\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n) \\ q \leftarrow \mathsf{Query}(\mathsf{ck}, i) \end{array} \right\}.$$

Then for every $\lambda, n \in \mathbb{N}$, and every $x \in \{0,1\}^n$:

$$\Pr\left[\begin{array}{c} \mathsf{Answer}^x(q) \\ \text{probes the } i\text{th bit of } x \end{array} : \begin{array}{c} i \xleftarrow{R} [n] \\ q \xleftarrow{R} D_i \end{array}\right] < T/n + \mathrm{negl}(\lambda). \tag{20}$$

Proof. Let $\mathcal{A}$ be an algorithm, that given an online query q, runs $\mathsf{Answer}^x(q)$ and outputs an index $i \in [n]$ chosen uniformly at random from the set of database indices that Answer probes. Then

$$\Pr\left[\mathcal{A}(q) = i : \begin{array}{c} i \xleftarrow{R} [n] \\ q \xleftarrow{R} D_i \end{array}\right] \geq \frac{1}{T} \cdot \Pr\left[\begin{array}{c} \mathsf{Answer}^x(q) \\ \text{probes the } i\text{th bit of } x \end{array} : \begin{array}{c} i \xleftarrow{R} [n] \\ q \xleftarrow{R} D_i \end{array}\right].$$

From the security of the PIR scheme and Proposition 26, we have that for every efficient algorithm $\mathcal{A}$, it holds

$$\Pr\left[\mathcal{A}(q) = i : \begin{array}{c} i \xleftarrow{R} [n] \\ q \xleftarrow{R} D_i \end{array}\right] \leq \frac{1}{n} + \mathrm{negl}(\lambda),$$

and therefore

$$\Pr\left[\begin{array}{c} \mathsf{Answer}^x(q) \\ \text{probes the } i\text{th bit of } x \end{array} : \begin{array}{c} i \xleftarrow{R} [n] \\ q \xleftarrow{R} D_i \end{array}\right] \leq T \cdot \Pr\left[\mathcal{A}(q) = i : \begin{array}{c} i \xleftarrow{R} [n] \\ q \xleftarrow{R} D_i \end{array}\right] \leq \frac{T}{n} + \mathrm{negl}(\lambda)\,.$$

□

The crux of the proof of Theorem 23 is the following lemma, which implies that a good offline/online PIR scheme implies a good solution to Yao's Box Problem:

Lemma 50. *If there exists a PIR scheme such that, on security parameter $\lambda \in \mathbb{N}$ and database size $n \in \mathbb{N}$,*

- *the client downloads at most C bits in the offline phase,*
- *the server probes at most T bits of the database in the online phase, and*
- *the client obtains its desired bit of the database with probability at least $1/2 + \epsilon_{\mathsf{pir}}$,*

then there exists an algorithm that ϵ_{box}-solves Yao's Box Problem on parameter n using C bits of advice with T queries, for $\epsilon_{\mathsf{box}} \geq \epsilon_{\mathsf{pir}} - T/n - \mathrm{negl}(\lambda)$.

Proof. We construct an algorithm $(\mathcal{A}_1, \mathcal{A}_2)$ that solves Yao's Box Problem with the stated parameters. Let $\Pi = (\mathsf{Setup}, \mathsf{Hint}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Reconstruct})$ be an offline/online PIR scheme. By Lemma 49 and an averaging argument, there exists a choice of the PIR client's random coins such that (20) holds when running the client with that specific choice of random coins. Hardcode this set of random coins into $\mathcal{A}_1$ and $\mathcal{A}_2$. (Algorithms for Yao's box problem can run in unbounded time, so even a uniform algorithm can find this good set of random coins via brute-force search.)

- **Preprocessing.** The algorithm $\mathcal{A}_1$, given access to an oracle $\mathcal{O} : [n] \to \{0,1\}$, executes the following steps:
  - Construct the string $x \leftarrow \langle \mathcal{O}(1), \cdots, \mathcal{O}(n) \rangle \in \{0,1\}^n$.
  - Using the hardcoded set of random coins, run $(\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n)$.
  - Then, run $h \leftarrow \mathsf{Hint}(x, q_h)$.
  - Output h as the advice string $\mathsf{st}_{\mathcal{O}}$.
- **Online.** The algorithm $\mathcal{A}_2$, given an index $i \in [n]$ and access to an oracle $\mathcal{O} : [n] \to \{0,1\}$, executes the following steps:
  - Using the hardcoded set of random coins, run $(\mathsf{ck}, q_h) \leftarrow \mathsf{Setup}(1^\lambda, n)$.
  - Compute $q \leftarrow \mathsf{Query}(\mathsf{ck}, i)$.

- Run $a \leftarrow \mathsf{Answer}^x(q)$. Whenever the algorithm needs to read the jth bit of x, compute this bit as $x_j \leftarrow \mathcal{O}(j)$. If the algorithm every queries for bit i, output "0."
- Output $\mathsf{Reconstruct}(\mathsf{ck}, h, a)$.

By construction, it holds that:

- algorithm $\mathcal{A}_1$ outputs an advice string of at most C bits, since Hint outputs a string of at most C bits,
- algorithm $\mathcal{A}_2$ makes at most T queries, since the algorithm Answer probes at most T bits of the database, and
- algorithm $\mathcal{A}_2^{\mathcal{O}}(\cdot, i)$ never queries its oracle on point i.

By Lemma 49 and our choice of the algorithm's random coins, we know that the online algorithm halts early with probability at most $T/n + \mathrm{negl}(\lambda)$ over the random choice of $i \xleftarrow{R} [n]$. Conditioned on the algorithm not halting early, the PIR correctness property implies that $\Pr[\mathcal{A}_2^{\mathcal{O}}(\mathcal{A}_1^{\mathcal{O}}(), i) = \mathcal{O}(i)] \geq 1/2 + \epsilon_{\mathsf{pir}}$. Therefore, algorithm $(\mathcal{A}_1, \mathcal{A}_2)$ ϵ_{box}-solves the Box Problem for $\epsilon_{\mathsf{box}} \geq \epsilon_{\mathsf{pir}} - T/n - \mathrm{negl}(\lambda)$. □

The proof of the main Theorem now follows: by Lemma 50 there exists an algorithm for Yao's Box Problem achieving $\epsilon_{\mathsf{box}} \geq \epsilon_{\mathsf{pir}} - T/n - \mathrm{negl}(\lambda)$. Since $\epsilon_{\mathsf{box}} \leq \widetilde{O}(\sqrt{C(T+1)/N})$ by Theorem 48, we get

$$\epsilon_{\mathsf{pir}} \leq \epsilon_{\mathsf{box}} + \frac{T}{N} + \mathrm{negl}(\lambda) \leq \widetilde{O}\left(\frac{T}{N} + \sqrt{\frac{C(T+1)}{N}}\right) + \mathrm{negl}(\lambda).$$

G An alternative abstraction: Sparse distributed point functions

In this subsection, we show that it is possible to view our PIR schemes as being constructed from a new tool we call *sparse distributed point functions* ("sparse DPFs"), inspired by the standard notion of DPFs [GI14]. In the following discussion, let $e_i \in \mathbb{F}^n$ denote the vector of zeros in $\mathbb{F}^n$ with a "1" at index i.

Standard DPFs. Intuitively, a DPF gives a way to produce two compressed additive shares of a vector $e_i \in \mathbb{F}^n$, for any $i \in [n]$. While an information-theoretically secure sharing of e_i must have shares of length $\Omega(n)$, a surprising line of work [GI14, BGI15, BGI16] shows that under the minimal assumption that one-way functions exist, it is possible to construct two computationally hiding shares of e_i that have size $O_\lambda(\log n)$. This construction immediately implies two-server computational PIR schemes with total communication $O_\lambda(\log n)$.

A standard DPF defined over a finite field $\mathbb{F}$ consists of two algorithms $(\mathsf{Gen}, \mathsf{Eval})$ where:

- $\mathsf{Gen}(1^\lambda, n, i) \to (k_{\mathsf{left}}, k_{\mathsf{right}})$ takes as input a security parameter λ, a dimension $n \in \mathbb{N}$, and a value $i \in [n]$, and outputs two keys k_{left} and k_{right}.
- $\mathsf{Eval}(k) \to v \in \mathbb{F}^n$ takes as input a key and outputs a vector $v \in \mathbb{F}^n$.

A DPF satisfies two properties:

– **Correctness.** For all $\lambda \in \mathbb{N}$, polynomially bounded $n = n(\lambda)$, and $i, j \in [n]$,

$$\Pr[\mathsf{Eval}(k_{\mathsf{left}}) + \mathsf{Eval}(k_{\mathsf{right}}) = e_i : (k_{\mathsf{left}}, k_{\mathsf{right}}) \leftarrow \mathsf{Gen}(1^\lambda, n, i)] > 1 - \mathrm{negl}(\lambda).$$

– **Security.** There exists an efficient simulator $\mathsf{Sim}_{\mathsf{right}}$ such that for all polynomially bounded $n = n(\lambda)$ on security parameter $\lambda \in \mathbb{N}$, and all $i \in [n]$,

$$\{k_{\mathsf{right}} : (k_{\mathsf{left}}, k_{\mathsf{right}}) \leftarrow \mathsf{Gen}(1^\lambda, n, i)\} \approx_c \{\mathsf{Sim}_{\mathsf{right}}(1^\lambda, n)\},$$

and there exists an analogous simulator $\mathsf{Sim}_{\mathsf{left}}$.

Ideal tools we cannot construct. For our application, we would like to construct distributed point functions in which both of the vectors $\mathsf{Eval}(k_{\mathsf{left}}) \in \mathbb{F}^n$ and $\mathsf{Eval}(k_{\mathsf{right}}) \in \mathbb{F}^n$ (for keys k_{left} and k_{right}) are *sparse*. This would immediately give computational two-server PIR schemes in which the servers run in sublinear time. Unfortunately, such a construction is impossible. (For one, it contradicts the $\Omega(n)$ PIR server time lower bound of Beimel et al. [BIM04].)

Since that initial goal is too ambitious, we could ask for something more limited—a DPF scheme in which

– it is possible to sample the left key before choosing the index $i \in [n]$,
– the evaluation of the right key $\mathsf{Eval}(k_{\mathsf{right}}) \in \mathbb{F}^n$ is sparse.

Such a DPF construction would immediately give rise to an offline/online PIR scheme with communication complexity proportional to the size of the DPF keys.

Unfortunately, even if we impose only the first restriction (that it is possible to sample left key before choosing the hidden index $i \in [n]$), the only known construction of this primitive is from private constrained PRFs [BLW17], which in turn require multilinear maps—a very heavy cryptographic assumption.

A useful relaxation: Sparse DPFs. Our idea is to construct a relaxation of distributed point functions that:

(a) we can build from one-way functions (or even without assumptions) and that
(b) suffices for offline/online PIR schemes with sublinear online time.

Informally, the key-generation routine Gen for a sparse DPF takes as input only the security parameter λ and dimension n (*not* the hidden index i) and outputs a *family* of DPF keys $K_{\mathsf{left}} = (k_1, \ldots, k_m)$. The guarantee of a sparse DPF is that on dimension n, there exists an efficient routine Choose that takes as input K_{left} and any index $i \in [n]$, and samples keys k_j and k_{right} such that, as in a standard DPF:

– $\mathsf{Eval}(k_j) + \mathsf{Eval}(k_{\mathsf{right}}) = e_i \in \mathbb{F}^n$ and
– k_{right} computationally hides the special index i,

but also such that:

– $k_j \in K_{\mathsf{left}}$ and
– $\mathsf{Eval}(k_{\mathsf{right}}) \in \mathbb{F}^n$ is sparse.

It turns out that these properties together are enough to construct an offline/online PIR scheme with sublinear online time, as long as the keys are short

enough and $\mathsf{Eval}(k_{\mathsf{right}}) \in \mathbb{F}^n$ is a sufficiently sparse vector. To sketch how to build an offline/online PIR scheme from a sparse DPF:

- In the offline phase, the client runs $K_{\mathsf{left}} = (k_1, \ldots, k_m) \leftarrow \mathsf{Gen}(1^\lambda, n)$ and sends K_{left} to the left server. For each $j \in [m]$, the left servers sends the hint value $h_j \leftarrow \langle x, \mathsf{Eval}(k_j) \rangle \in \mathbb{F}$ to the client, where we view the database x as a vector in $\mathbb{F}^n$.
- In the online phase, the client runs $(j, k_{\mathsf{right}}) \leftarrow \mathsf{Choose}(K_{\mathsf{left}}, i)$ and sends k_{right} to the right server. The right server computes the answer $a \leftarrow \langle x, \mathsf{Eval}(k_{\mathsf{right}}) \rangle \in \mathbb{F}$ and returns the answer to the client. Notice that if $\mathsf{Eval}(k_{\mathsf{right}})$ is non-zero in $o(n)$ coordinates, the right server can compute the answer in sublinear time.

  The client computes its desired database entry $x_i \in \mathbb{F}$ as:

$$\begin{aligned} x_i &= h_j + a \\ &= \Big\langle x, \mathsf{Eval}(k_j) \Big\rangle + \Big\langle x, \mathsf{Eval}(k_{\mathsf{right}}) \Big\rangle \\ &= \Big\langle x, \mathsf{Eval}(k_j) + \mathsf{Eval}(k_{\mathsf{right}}) \Big\rangle \\ &= \langle x, e_i \rangle \in \mathbb{F}. \end{aligned}$$

Formally, a *sparse DPF* is parameterized by a sparsity $s : \mathbb{N} \to \mathbb{N}$ and a family size $m : \mathbb{N} \to \mathbb{N}$, and is a three-tuple of algorithms $(\mathsf{Gen}, \mathsf{Choose}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Gen}(1^\lambda, n) \to K_{\mathsf{left}} = (k_1, \ldots, k_m)$ takes as input a security parameter λ and a dimension $n \in \mathbb{N}$, and outputs a "left key," which is a tuple of $m = m(n)$ keys.
- $\mathsf{Choose}(K_{\mathsf{left}}, i) \to (j, k_{\mathsf{right}})$ takes as input a left key and an index $i \in [n]$, and outputs a value $j \in [m]$, and a right key k_{right}.
- $\mathsf{Eval}(k) \to v \in \mathbb{F}^n$ takes as input a key and an index $i \in [n]$ and outputs a vector $v \in \mathbb{F}^n$.

  A sparse DPF satisfies three properties:
- **Correctness.** For all $\lambda \in \mathbb{N}$, polynomially bounded $n = n(\lambda)$, and $i \in [n]$,

$$\Pr\left[\mathsf{Eval}\left((K_{\mathsf{left}})_j\right) + \mathsf{Eval}(k_{\mathsf{right}}) = e_i : \begin{array}{c} K_{\mathsf{left}} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ (j, k_{\mathsf{right}}) \leftarrow \mathsf{Choose}(K_{\mathsf{left}}, i) \end{array}\right] > 1 - \mathrm{negl}(\lambda).$$

- **Security.** There exists an efficient simulator $\mathsf{Sim}_{\mathsf{right}}$ such that for all polynomially bounded $n = n(\lambda)$ on security parameter $\lambda \in \mathbb{N}$, and all $i \in [n]$,

$$\left\{k_{\mathsf{right}} : \begin{array}{c} K_{\mathsf{left}} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ (j, k_{\mathsf{right}}) \leftarrow \mathsf{Choose}(K_{\mathsf{left}}, i) \end{array}\right\} \approx_c \{\mathsf{Sim}_{\mathsf{right}}(1^\lambda, n)\}.$$

  If these distributions are statistically close, we say that the sparse DPF has *s*tatistical security.
- **Sparsity.** For all $\lambda \in \mathbb{N}$, polynomially bounded $n = n(\lambda)$, and $i \in [n]$, $K_{\mathsf{left}} \leftarrow \mathsf{Gen}(1^\lambda, n)$ and $(j, k_{\mathsf{right}}) \leftarrow \mathsf{Choose}(K_{\mathsf{left}}, i)$, the vector $\mathsf{Eval}(k_{\mathsf{right}}) \in \mathbb{F}^n$ has at most $s = s(n)$ non-zero coordinates.

We can view our two-server offline/online PIR schemes as implicitly constructing a sparse DPFs.

Corollary 51 (Statistically secure sparse DPFs). *There exists a statistically secure sparse DPF scheme in which, on security parameter λ and dimension n, the key length, family size $m(n)$, and sparsity $s(n)$ are all $\widetilde{O}_\lambda(\sqrt{n})$.*

Corollary 52 (Computationally secure sparse DPFs). *If one-way functions exist, there exists a sparse DPF scheme as in Corollary 51 with the same efficiency parameters, except that on security parameter λ and dimension n, the right key size decreases to $\mathrm{poly}(\lambda, \log n)$ bits.*